A Study of Failure in a Distributed Pi-calculus

Adrian Francalanza

Submitted for the degree of D. Phil.

University of Sussex

May 2005

Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Signature:

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Matthew Hennessy, for inspiring me through out these four years with his advice, criticism and encouragement, and for making it an enjoyable learning experience. I am also indebted to Julian Rathke, for guiding me during the initial stages of my studies.

A special thanks goes to my wife, Juniper, and my children, Gaetano and (then unborn) Liliana, for helping me to keep things in perspective.

Another special thanks goes also to my mother and my younger sister, Helena, for their constant support throughout the four years. One final special thanks goes to my father, for forgoing his own doctorial studies so as not to uproot his two children, the eldest of whom had just started school. This D.Phil is yours as much as it is mine.

Abstract

In this thesis, we study the behaviour of programs distributed across a network, where the components making up the network, that is nodes and links, may fail. The foundational framework used throughout this study is that of Dπ, a language in which located processes may migrate between dynamically created locations to interact with other processes residing at the same location.

In the first part of the thesis, a behavioural theory for systems in the presence of both node and link failures is developed. The work is carried out at the level of configurations, which represent the distributed network over which system execute and allow individual nodes and links to fail. In addition, the language is extended by a new construct for detecting and reacting to these failures. A bisimulation equivalence between systems running over the same network is then defined, based on labelled actions; these labels record, in addition to the effect actions have on the systems, the effect on the actual state of the underlying network together with the view of this state known to observers. Subsequently, it is shown that the equivalence is fully abstract, that is two systems will be differentiated if and only if there is a computational context, consisting of a network and an observer, which can see the difference.

In the second part of the thesis, a definition of system fault tolerance is formalised. The definition relies on the notion of controlled fault injection on confined locations and any observed difference in behaviour at public locations as a result of the fault. The separation between public and confined locations is enforced through a type system for which a subject reduction property is proved. Sound tractable bisimulation techniques to determine when systems are fault tolerant are also developed.

In the third and final part of the thesis, the definition of fault tolerance and the corresponding bisimulation techniques are applied to formally prove properties about distributed algorithms. More specifically, a standard algorithm for solving consensus with perfect failure detectors is encoded into our framework; the three properties defining consensus, that is agreement, validity and termination, are also encoded into this framework. The encoded algorithm is first shown to satisfy our definition of fault tolerance and from this result, the necessary consensus properties of agreement, validity and termination are extracted.

Contents

1 Introduction **2**

1.1 Distribution of Code 2

1.2 Faults, Failure and Partial Failure 2

1.2.1 Faults 3

1.2.2 Failures 3

1.2.3 Failure Detection 4

1.3 Dependability 5

1.3.1 Fault tolerance 5

1.3.2 Replication 5

1.4 Distributed Process Calculi and Failure 6

1.5 Outline 10

1.5.1 Chapter Dependencies 11

1.5.2 Contributions of the Thesis 11

1.5.3 Preliminaries 11

2 Dπ with Fail-Stop Location Failure **13**

2.1 DπLoc syntax 13

2.2 The reduction semantics of DπLoc 15

2.2.1 Reduction barbed congruence 18

2.3 A bisimulation equivalence for DπLoc 23

2.4 Summary 28

3 Dπ with Location and Link Failure **30**

3.1 DπF syntax 30

3.2 Reduction Semantics of DπF 31

3.2.1 Reduction barbed congruence 35

3.3 A labelled transition system for DπF 36

3.4 A bisimulation equivalence for DπF 45

3.5 Full-Abstraction 48

3.5.1 Reduction barbed congruence for DπF 48

3.5.2 Soundness 52

3.5.3 Completness 59

3.6 Summary 67

4 Fault Tolerance **69**

4.1 Motivation 69

4.2 Typed DπLoc 73

4.2.1 Reduction Semantics 75
4.2.2 Equivalences in typed DπLoc 81
4.2.3 Bisimulation up-to proof techniques 90
4.3 Defining Fault Tolerance 95
4.4 Bisimulation techniques for Fault Tolerance 103
4.5 Summary 109

5 Case Study: Consensus **111**
5.1 Consensus Overview 111
5.2 Defining Consensus in typed DπLoc 113
5.3 Implementing the Consensus-solving Algorithm in typed DπLoc 115
5.4 A Correctness proof for the Consensus Algorithm using Intensional Tests 118
5.5 Analysis of the Consensus-solving Algorithm and Tests 124
5.6 Intentional Test Termination for the Consensus-Solving Algorithm 129
5.7 Summary 137

6 Conclusion and Outlook **139**
6.1 Results obtained 139
6.2 Future Work 140

A Notation **142**
A.1 DπLoc Notation 142
A.2 DπF Notation 143
A.3 Typed DπLoc Notation 145
A.4 Auxilliary Proofs 146

List of Tables

Table 1. Syntax of typed DπLoc 14
Table 2. Local Reduction Rules for DπLoc 16
Table 3. Network Reduction Rules for DπLoc 17
Table 4. Contextual Reduction Rules for DπLoc 17
Table 5. Structural Rules for DπLoc 18
Table 6. Operational Rules(1) for DπLoc 23
Table 7. Operational Rules(2) for DπLoc 24
Table 8. Operational Rules(3) for DπLoc 25

Table 9. Syntax of DπF 31
Table 10. New Network Reduction Rules for DπF 32
Table 11. New Structural Rules for DπF 33
Table 12. Network Operational Rules(2) for DπF 42
Table 13. Contextual Operational Rules(3) for DπF 43
Table 14. The derived lts for DπF 46

Table 15. Syntax of typed DπLoc 73
Table 16. Typing rules for typed DπLoc 76
Table 17. Runtime Error 77
Table 18. Local Operational Rules for Typed DπLoc 84
Table 19. Network Operational Ruled for typed DπLoc 84
Table 20. Context Operational Rules for typed DπLoc 85
Table 21. The derived lts for typed DπLoc 86
Table 22. β-Transition Rules for Typed DπLoc 90
Table 23. β-Equivalence Rules for Typed DπLoc 91
Table 24. Network Operational Ruled for typed DπLoc 103

Table 25. Consensus Algorithm in typed DπLoc 116
Table 26. Reduction and Operational Rules for monitor in typed DπLoc 117
Table 27. Requirement Contexts in typed DπLoc 119

Chapter 1

Introduction

This thesis presents a study of *system behavior* in the presence of *partial failure* caused by *location* and *link faults*. The study focuses on:

- defining reasonable *equivalences* for systems running over the same network and developing *theories* that facilitate proofs of system equivalence.
- establishing formal definitions of system *dependability* in response to partial failure and developing *theories* that facilitate proofs of system dependability.
- applying these formal definitions and theories to existing work in the field of distributed computing.

1.1 Distribution of Code

Location transparency is hard to attain over Wide Area Networks (WANs) [CG00], that is infrastructures that may span the globe such as the Internet, because:

- *administrative and security* concerns preclude unfettered computation across *domains* and as a result, *virtual locations* are induced to facilitate the control of computation.
- *physical constraints* such as *high communication latencies* and *unpredictable random failures* make it hard to conceal the underlying structure of code distribution.

As a result, a number of programming languages and calculi have arisen which describe the *location* were computation is executing, giving the programmer more control. In this thesis, we focus on a calculus that describes *physical* locations rather than *virtual* locations and study the behaviour of distributed code, that is *systems*, in the presence of changes in the state of locations and location structure.

1.2 Faults, Failure and Partial Failure

The literature often makes a distinction between *faults* and *failures*. In [Chr91], failure is defined as deviations in program behaviour, or *erratic behaviour*, caused by faults, defined

as defects at the lowest level of abstractions. Our study is based on *two-tiered* terms called *configurations* which have the following form:

$$\mathsf{Network} \triangleright \mathsf{System}$$

The first tier, Network, is a representation of the state of the network whereas the second tier, System, represents distributed code. In configurations, the behaviour of the code in System is always defined with respect to Network, capturing the intuition that the code System is *executing over* the network defined in Network. In this setting, faults are represented at the first tier, Network, as constraints that effect the execution of the code in the second tier, System, where we represent failure. Since the code in System is distributed, faults only affect part of its behaviour, a phenomenon often termed as *partial failure*.

1.2.1 Faults

In general, faults may be either of two kinds:

Permanent Faults Once a fault occurs, there is no way to recover the previous state of the network.

Transient Faults If a network incurs a transient fault, it can restore back to its original state through *fault repair*.

In our study, we limit ourselves to permanent faults. In particular we consider permanent node faults and permanent link faults which may occur *dynamically* during the computation; the state of Network changes by executing *fault injecting* commands at the System level. We stress that these fault injecting commands are not intended as programming constructs but rather as a mechanised methods how to change the state of Network at runtime.

1.2.2 Failures

As already stated, faults in a distributed setting lead to partial failure, where the nodes themselves act as *units of failure*. There are various failure classifications in the literature. Coulouris *et al.* [CDT01] identify three failure categories in a distributed setting:

Omission Failures These failures refer to migrating messages from one location and the other which may be delivered incorrectly, lost or corrupted.

Link Failures These failures refer to code that gets *disconnected* from other code and may potentially become *unreachable*.

Node Failure These failures refer to code limited to a particular location that typically *stops executing* or *behaves abnormally.*

Another classification of failure, due to [LSP82, SS83] is based on two categories:

Fail-stop All entities affected by the fault, stop executing and any volatile storage is lost.

Byzantine Entities affected by the fault may lose part of the volatile storage, but continue executing, exhibiting abnormal unconstrained behaviour.

We here study *node* and *link failure* referring to erratic behaviour caused by node faults and link faults respectively. The node failure we consider is strictly fail-stop: as soon as a fault happens in a node,all code residing at that node stops executing. The link failure we describe (occasionally referred to as permanent *connectivity failure*) can be envisaged as fail-stop since, as soon as a fault occurs in a link connecting two nodes, the link stops functioning and no code can migrate directly between the two disconnected nodes. Alternatively, this link failure can also be envisaged as byzantine, since the code affected by the link fault, typically the code at the two nodes connected by the link, does not stop executing, but rather behaves abnormally in the form of complete loss of code during migration. Such behaviour is however not purely byzantine because is not arbitrary: a faulty link *never* allows any code to migrate through it. Because of this reason too, we do not describe any form of omission failure in our study.

1.2.3 Failure Detection

Faults may be *observed* in two ways from the System tier: either through failure, as we have already discussed, or through *failure detection*. The latter is often performed by observing faults at the lower level and subsequently inferring potential failure caused by the fault. In asynchronous networks such as WANs, it is generally accepted that faults, and thus failures, are hard to detect [FLP85]: high latencies preclude tight synchronisations between nodes and the non-uniformity of WAN links and nodes makes it hard to set communication bounds and estimate CPU response times. Thus, it is hard to distinguish between an unreachable node and a reachable node with either slow CPU response time or slow connection.

In practice however, *unreliable* failure detection is often employed, using mechanisms such as acknowledgments and time-outs to establish an *approximation* to whether a node is reachable or not [MAB+98]. Toueg and Chandra [CT96], give a classification of *failure detectors*, defined as local separate modules that output a list of suspected failed entities. The classification is based on two criteria:

Completeness If a failure detector is complete, it suspects *all* the failed entities.

Accuracy If a failure detector is accurate, it *never* suspects an entity that has not failed.

In our study, failure detection is performed through a branching ping construct that tests for the *accessibility* of a remote location from the location where it is executing. Even though this construct does not adhere strictly to the definition of a failure detector - it is not a separate module in itself and does not output a list of suspected failed entities - we argue that it is *accurate* in the sense that it never suspects that a location is unreachable unless it *is* unreachable. Using this construct, we also implement failure detection mechanisms (called monitors) that are *complete* in the sense that when a location becomes unreachable, the mechanism will eventually detect the failure. We consciously overload the definitions

of [CT96] and refer to this single monitor mechanism as a failure detector, even though it only tests for a single location; a proper failure detector can be constructed from an array of such mechanisms. This allows us to postulate that our language uses a specific class of failure detectors called *perfect* failure detectors, denoted in [CT96] as $\mathcal{P}$. Even though this level of failure detection may be considered unrealistic for a WAN setting, it still proves to be an adequate level of abstraction for studying the interplay between node and link faults and the respective observation at the distributed program level.

1.3 Dependability

One reason for the study of programs in the presence of failure is to be able to construct more *dependable* systems, that is, systems exhibiting a high probability of *behaving* according to their *specification*. System dependability is often expressed through attributes like maintainability, availability, safety and *reliability*, the latter of which is defined as a measure of the *continuous delivery* of *correct behaviour*, [VR01]. In our setting, there are a number of approaches for achieving system dependability in the presence of faults, ranging from fault removal, fault prevention and *fault tolerance*.

1.3.1 Fault tolerance

The fault tolerant approach to system dependability consist of various techniques that employ *redundancy* to prevent faults from generating failure: two forms of redundancy are *space redundancy*, that is using several copies of the same system components and *time redundancy*, that is performing the same chunck of computation more than once. Redundancy can also be managed in various ways: certain fault tolerant techniques are based on *fault detection* which subsequently trigger *fault recovery*. If enough redundancy is used, fault recovery can lead to *fault masking*, where the specified behaviour is preserved despite the fault. Other techniques however do not use fault detection and still attain fault masking. However, these techniques tend to be more expensive in terms of redundancy usage; they often use a brute force approach where both time and space redundancy are used at the same time.

1.3.2 Replication

In distributed programs, redundancy often takes the form of *replication*, that is *space* redundancy spread across multiple locations. In general, the higher the replication, the higher the redundancy and thus the greater the potential for fault tolerance. Nevertheless, fault tolerance also depends on how redundancy is managed. There are various ways to manage replication and one such classification, due to [VR01], is:

Active Relication: It uses both space and time redundancy, where *all* the replicas are invoked by a coordinator, they *all* compute the same computation which is then aggregated by the coordinator. If the replicas carry a notion of state, the coordinator needs to ensure that all the replica states are in synchrony through techniques such as sequencing of multiple invocations.

Passive Replication: It uses one *primary* replica and a number of *secondary* replicas. Passive replication is still based on space redundancy but tries to minimise time redundancy by invoking only the primary replica; the secondary (passive) replicas are invoked only when the primary replica fails. If the replicas carry a notion of state, the secondary replicas have to periodically checkpoint and synchronise their state with that of the primary replica.

Lazy Replication: It lies half way between active and passive replication. The main concept is a separation between operations that invoke the replicas. If an operation changes the state of the replica, often referred to as a *write* operation, it is sent to all replicas in active replication fashion. If however the operation does not alter the state, that is a *read* operation, only the primary replica is invoked. In essence, lazy replication tries to obtain the best of both worlds and circumvent checkpoints of replication state.

In our studywe limit ourselves to examples with replicas that do not encode a notion of state, thus referred to as *stateless* replicas in this test. As a result, the third type of replication discussed about, that is lazy replication, collapses into either of the first two types (active and passive), since there is no distinction between write and read operations. Despite of this simplification, we argue that the examples considered still capture the essence of fault tolerant computation through replication. One can further postulate that the theory developed is general enough to be applied to examples with replicas that encode notions of state.

1.4 Distributed Process Calculi and Failure

Our study is specifically concerned with developing bisimulation techniques for distributed calculi which describe notions of partial failure. These bisimulations are justified using contextual equivalences such as those defined by Honda and Yoshida [HY95] and Milner and Sangiorgi [MS92, San92, SW01]. We follow the work of Hennessy, Rathke and Merro [HR04, HMR04] who adapted reduction barbed congruence to the typed π-calculus and to a distributed π-calculus called Dπ [HR02], and gave bisimulations that coincide with these congruences; throughout this thesis, we will extend these results to an adaptation of Dπ that describes various notions of failure.

The first use of process calculi to study failure, redundancy and fault tolerance is due to K. V. S. Prasad in [Pra87]. The calculus used for this study is a variant of CCS, [Mil89], that is neither distributed nor expresses failure directly; instead failure is encoded using a specially labelled action f and guarded choice. The encoding also leads to a very thigh synchronisation between failure and failure detection, which is encoded as the co-action of f. Most of the work in [Pra87] targets two aims: the formulation of operators which allow an easy encoding of failing and restarting processes (called displace, audit and checkpoint operators), and more importantly, the development of proof techniques that deal with the large number of states generated as a result of the interleaving of failure. The main result of the work is the *Synchronised Displacement Theorem*, which allows an algebraic manipulation of terms and can be used to prove that systems are, in some sense, fault tolerant with respect to a specification. A difference between our work

and Prasad's is that we address the problem of the increased number of states through the interleaving of failure differently, using up-to techniques to abstract over confluent interleaving. Despite of these differences, this work is related to ours because, to the best of our knowledge, it is the first time concepts such as "failure-free behaviour", redundancy (called "duplication"), "synchronisation" between replicas (called duplicates) and more generally fault tolerance are identified within the field of process calculi; our study happends to address all of these concepts.

More specific to our study, failure in a *distributed setting* has already been studied from the point of view of process calculi. The various efforts vary with respect to

- the structure of locations assumed
- the type of failure associated with this structure
- restrictions for distributed code
- methods for failure detection.

We here give a general, but not necessarily exhaustive, overview of this body of work, to help the reader relate our study to other work in the field.

The distributed CCS The work carried out by Hennessy and Riely, [RH01], was the starting point of the first calculus presented in this thesis called DπLoc. In [RH01], processes are distributed across a *fixed* number of *named* locations, organised in a *flat* structure where locations are killed in *fail-stop* fashion, without affecting, or depending on, the state of other locations. The distribution of processes in this calculus does not restrict computation: two processes can interact freely, irrespective of the location where they reside. Thus, locations merely act as units of failure. This failure dependency between processes can be altered during computation through migration from one location to another. Failure detection is performed through a *perfectly accurate* conditional construct, similar to the ping construct used in our study, which branches according to the status of the tested location. Inspite of the similarities, there are important differences between [RH01] and our closest work, DπLoc. In our calculus, locations also denote restrictions in computation since we prohibit distributed communication: More specifically communication between distributed processes has to be decomposed into two atomic steps, that is migration and *local* communication. Most importantly, our calculus is value passing, whereas the processes [RH01] are not[1]; we also allow the creation of fresh locations. Because of these enhancements, the theory for our calculus has to deal with scope extrusion of location names.

The π_l-calculus Our work was also influenced by the seminal work on process calculi and failures, the π_l-calculus [AP94, Ama97], by Amadio and Prasad. Similar to our first calculus, DπLoc, in the π_l-calculus processes are distributed across *named* locations, organised in a *flat* structure, which may die in *fail-stop* fashion, just like in DπLoc. Direct

[1] They are essentially CCS processes.

distributed computation is also prohibited; processes are allowed to *spawn* remote processes and create new locations. The π_l-calculus also assumes *unicity of receptors*, meaning that inputs on a particular channel can only be performed at a unique location: this permits asynchronous messages to be implicitly forwarded to the destination location in the calculus, where a local communication can occur. Viewed otherwise, locations in π_l-calculus not only represent units of failure for processes, but also boundaries on where communications on certain channels can occur. The π_l-calculus processes can also perform failure detection through pinging, a *perfectly accurate* conditional construct similar to the one used in DπLoc. Despite of the comonalities between the π_l-calculus and DπLoc, the theory developed for these two calculi is quite different: in [AP94, Ama97] they give an encoding of the π_l-calculus into a location-less π-calculus and then propose to use the reasoning tools developed for the location-less calculus for the encoded terms; we develop sound an complete bisimulation techniques in the source language directly.

The Timed distributed π-calculus Another distributed value-passing process calculus studying failure was developed by Berger and Honda, [HB00, Ber04], with the ultimate aim of encoding and formally proving atomicity for the two-phase commit protocol. The π-calculus processes in this calculus are distributed across a *fixed* set of *unnamed* locations where every location is associated to a set of *access points* instead of a location name; a *flat* structure is imposed on these locations once again. The access points denote channel names where the respective inputs may occur. Once again, *unicity of receptors* for all locations is assumed here, permitting a location to be *uniquely identified* through its access points. The calculus *does not* allow processes to migrate, but instead, allows distributed computation across locations. Two kinds of failure are modelled in this calculus: the first is *message loss* for distributed computation; the second is *transient location failure*, whereby locations may fail and re-start. Transient failure in this calculus amounts to more than a mere pause and restart, since all the processes at a failed location are lost upon re-start. Apart from units of failure, locations also represent units of persistent storage: while a location is alive, processes may checkpoint processes that are launched as soon as a location is revived from failure. In [HB00], locations also denote a local clock, synchronising all the processes at that location: intuitively, every reduction step carried out by processes at a location denotes a unit of time passed at that location. Failure detection is carried out using timeouts, constructs that launch processes after a certain amounts of local time units (reductions) have passed. The failure detectors that can be built with the timeout primitive are not accurate, reflecting impossibility results of [FLP85], in a setting of asynchronous failing clocks. This added expressivity however, complicates the formulation of the subsequent theory of the calculus.

The Consensus Distributed Process Calculus Another tailor-made calculus, specifically developed for a formal understanding of a particular distributed problem is that of Nestmann, Fuzzati and Merro, [NFM03], targeting *distributed consensus*. In this calculus, processes are distributed across *enumerated* locations, organised in a *flat* structure. Based on the definition of distributed consensus, the failure modelled in this calculus is *fail-stop*

location failure. Due to its specific purpose, locations in this calculus simply denote units of process failure - stated otherwise, distributed communication *across location* is permitted. one main emphasis of this calculus is that of failure detection, which is faithful to the classification of [CT96], discussed earlier in § 1.2.3. More specifically, locations start in a *live, suspectable* state and can transit to either a *dead* state or an *immortal, unsuspectable* state. Failure detection is not carried out through any specialised construct; instead it is encoded within the language as asynchronous messages of the form "suspect *j*", triggered only if location *j* is in a live, suspectable state or dead state. This model of failure detection maps directly to the eventually weak accurate, strongly complete class of failure detectors, $\diamond S$, defined in [CT96]. Even though they give a semantics of this language, the study and correctness proofs of the consensus algorithm encoded in this language is carried out in terms of an abstract interpretation called a global view-matrix. They provide an encoding and inverse encoding of a subset of the calculus in terms of this matrix. This differs from the correctness proofs we present in our study for the consensus algorithms, carried out in the source language itself, using bisimulation techniques.

The distributed join-calculus There is also work that studies failure in a setting where the structure imposed on locations is not flat: the Distributed Join-calculus, [FGLD96], by Fournet, Gonthier, Levy, Maranget and Remy, studies failure of *uniquely named* locations organised as *forests*, that is a collection of *hierarchical* locations. This structure is also *dynamic*, in the sense that whole subtrees can move from one parent location to any other location; thus locations act as units of migration in addition to units of failure. The novel aspect of this calculus from our point of interest, is that they express fail-stop location failure with *hierarchical failure dependencies*: this means that when a location fails, all of its children locations in the hierarchy fail as well. Processes distributed across locations communicate by the implicit forwarding of asynchronous messages: once again, for this implicit forwarding to work properly, unicity of receptors is assumed. Location failure detection is also permitted in this language through a high level construct very close to a failure detector, which monitors a location and releases an asynchronous message as soon as the location being monitored fails. In [FGLD96], they give a reduction semantics for their language, but they do not develop a theory that allows reasoning about Distributed Join-calculus terms.

The tKlaim-calculus Another calculus were locations are not organised in a flat structure is the tKlaim calculus, or topological Klaim, by De Nicola, Gorla and Pugliese, [DNGP04]. As the name implies, locations are organised in a *dynamic* graph structure, where localised linda-like processes can freely reconfigure this graph structure by executing connection and disconnection commands at will. Despite the fact that the location graph structure is similar to that of DπF, our second calculus describing link failure, the details are different from ours: tKlaim does not model failure, firstly because computation does not behave erratically when two nodes are disconnected, but merely blocks until the connection is re-established, and secondly, because the state of the graph location structure before and after a disconnection is re-established are identical. In contrast, disconnections in DπF

are *permanent* and migrating code *is lost* when it asynchronously attempts to migrate to an accessible node. Correspondingly, no failure detection mechanism is provided in the tKlaim calculus. They do however, develop a theory for this distributed calculus with dynamic disconnections.

1.5 Outline

In this thesis we study system behaviour in the presence of both node and link failure and develop sound and complete bisimulation techniques to reason about systems executing in this setting. We also develop a theory of fault tolerance, that can be used to ensure that a system still satisfies its intended behaviour despite of present or future faults. We finally show how the theory can be applied to reason about distributed algorithms. We expect that our study will shed more light on the foundations of distributed computing in the presence of failure.

The body of the thesis is split into four main chapters. Chapter 2 introduces a distributed π-calculus called DπLoc, based on Dπ, [HR02, HMR04], where *fail-stop* location failure is expressed. The language features and reduction semantics are similar to [AP94, Ama97]: communication across locations is prohibited, processes are explicitly routed during migration, new locations can be created and killed at runtime, and failure may be observed through a perfectly accurate ping construct. The theory developed and result obtained are however different from Amadio and Prasad's work: we develop an lts and a weak bisimulation in the source language and prove that this bisimulation is sound and complete with respect to a reduction barbed congruence defined for systems running over the same network. This chapter lays the foundations for the following two Chapters.

Chapter 3 studies the behaviour of distributed code in the presence of both *permanent node* and *link failure*. We extend the calculus introduced in Chapter 2 so that network representations describe the connections between the nodes, new locations specify the locations they are connected to and a new construct for injecting link failures at runtime is introduced; the new calculus is called DπF. A novel aspect of DπF is that the perfectly accurate ping construct used for failure detection (carried over from the previous chapter), now yields *accessibility* information rather than liveness information about locations. Link failures complicate the notion of scope extrusion of location names and we discuss at length why the theory for this new language is not a trivial extension of the theory developed in the preceding chapter. At this point, we also justify the need for an extended formulation of a network representation that encodes the partial view of the observer. Based on this extended network representation, we define a *derived lts*, suppressing information from the labels based on the observer's partial view and show that the weak bisimulation, defined over the derived lts, coincides with the reduction barbed congruence defined for systems running over networks with link failure.

Chapter 4 addresses dependability, a fundamental aspect of behaviour in the presence of failure. Since failure are generally hard to prevent or repair, we focus on *fault tolerance* techniques for guaranteeing dependability of distributed software. We revert back to the

first calculus, DπLoc, and formalise a definition of fault tolerance based on the notion of a partially observable term and a controlled injection of faults in the non-observable part of the term. We develop a typed version of the language DπLoc, with a type system that guarantees that these two views, the observable and hidden views, are preserved during computation. After formulating a fault tolerance definition for well-typed DπLoc terms, we apply bisimulation techniques and develop up-to techniques to facilitate proofs showing that certain terms satisfy our fault tolerance definitions.

Chapter 5 is a case study where the theory developed in the preceding chapter is used to prove the correctness of a *consensus* algorithm. We specify consensus by encoding its conditions as a series of wrapper DπLoc contexts; we then encode the algorithm in DπLoc as well. Instead of proving the correctness of the algorithm directly with respect to the consensus requirements, we formulate the algorithm's behaviour as a fault tolerance problem, prove it using the theory of Chapter 4, and then extract the necessary results from the fault-tolerance properties proved.

1.5.1 Chapter Dependencies

Chapters 3, 4 and 5 contain the key proofs of our study. The material in the latter two chapters is independent of Chapter 3 and can be fully understood without reading the material of Chapter 3. Similarly Chapter 3 can be read independently of Chapters 4 and 5.Chapter 5 should be read in conjunction with Chapter 4, since the former relies heavily on the theory developed in the latter. Most of the concepts underpinning the theory of Chapter 3 and Chapter 3 are first introduced in a simpler setting in Chapter 2 and the reader should benefit by reading Chapter 2 before embarking on the more demanding Chapters.

1.5.2 Contributions of the Thesis

The main contributions of the thesis are found in Chapters 3 and 4. To the best of our knowledge, this is the first time permanent link failure has been studied from a point of view of a distributed process calculus. As far as we know, this is also the first time a definition of fault tolerance is formalised for a process calculus; apart from the work by K.V.S. Prasad, [Pra87], our work is the first one using *distributed* process calculi to study fault tolerance in a distributed setting. Even though process calculi have been previously used to prove correctness of distributed algorithms, we also believe that this is the first time bisimulation techniques are used to show the correctness of a consensus-solving algorithm.

1.5.3 Preliminaries

The reader is assumed to be knowledgeable of process calculi, in particular of the π-calculus: familiarity is assumed with a reduction semantics presentation of a calculus, comprising of reduction and structural rules. The reader is also assumed to be conversant with inductive and co-inductive proofs: the notion of weak bisimulations, defined over a labelled transition system with an internal action is used extensively in this work. The

interested reader is referred to [SW01] for the background knowledge required about bisimulation and up-to bisimulation techniques for the π-calculus; these ideas are extended and adapted to a distributed setting with failure in this work.

Chapter 2

Dπ with Fail-Stop Location Failure

In this section we generalise the work of [RH01] and present an extension from distributed CCS to Dπ called DπLoc, where *fail-stop* location failure can be expressed. We acquaint the reader with the syntax of the calculus, define a reduction semantics and adapt a reduction barbed congruence for Dπ that allows us to compare DπLoc terms. We also define an lts and a bisimulation for this calculus and state the main result of the chapter, that is that the bisimulation characterises the reduction barbed congruence.

2.1 DπLoc syntax

The syntax of DπLoc is given in Table 1 and assumes a set of *variables*, VARS, ranged over by $x, y, z, \ldots$, and a separate set of *names*, NAMES, ranged over by $n, m, \ldots$. This latter set is divided into locations, LOCS, ranged over by $l, k, \ldots$ and channels, CHANS, ranged over by $a, b, c, \ldots$. We use $u, v, \ldots$ to range over the set of *identifiers*, consisting of variables and names. When new names are created, they have associated with them a type, indicating whether they are to be used as a channel, $\mathtt{ch}$, or as a location, $\mathtt{loc[S]}$ with state $\mathtt{S}$, which can either be alive,$\mathtt{a}$, or dead, $\mathtt{d}$. A priori, there is not much sense in declaring a dead location, but the presence of this construct will facilitate the definition of the reduction semantics.

There are three syntactic categories in DπLoc. The first, *local processes* ranged over by P, Q, includes the standard π-calculus constructs for communication, $a!\langle V\rangle.P$ and $a?(X).P$, replicated input, $*a?(X).P$, name restriction $(\nu\, n : \mathtt{T})P$, where $\mathtt{T}$ types n as a channel or a location name, comparison $\mathsf{if}\ v = u.P\lceil Q\rceil$, inaction, $\mathbf{0}$, and parallel composition, $P|Q$. The values transmitted as part of a communication, ranged over by V, consist of tuples of identifiers. When input on a channel, names are deconstructed using patterns, ranged over by X; patterns are simply tuples of variables, each having a unique occurrence.

An important extension to Dπ is a programming construct which allows processes to react to *perceived faults* in the underlying communication network. In addition to the migration construct $\mathsf{go}\ l.P$, [HR02], we add a testing construct, $\mathsf{ping}\ l.P\lceil Q\rceil$, inspired from [AP94, Ama97, RH01]. This constructs acts as a conditional, based on the *perceived state*

Table 1. *Syntax of typed DπLoc*

Types

$$\mathtt{T},\mathtt{U} \;::=\; \mathtt{loc[S]} \mid \mathtt{ch} \qquad\qquad \mathtt{S},\mathtt{R} \;::=\; \mathtt{a} \mid \mathtt{d}$$

Processes

$$\begin{array}{rcllllll} P,Q & ::= & u!\langle V\rangle.P & \mid u?(X).P & \mid *\,u?(X).P & \mid \mathsf{if}\ v{=}u.P\lceil Q\rceil & \mid P|Q \\ & \mid & (\nu\, n\!:\!\mathtt{T})P & \mid \mathsf{go}\ u.P & \mid \mathsf{kill} & \mid \mathsf{ping}\ u.P\lceil Q\rceil & \mid \mathbf{0} \end{array}$$

Systems

$$\begin{array}{rclll} M,N,O & ::= & l[\![P]\!] & \mid N|M & \mid (\nu\, n\!:\!\mathtt{T})N \end{array}$$

of the location l; thus if l is reachable, process P is launched, otherwise Q is launched.

We also forgo the full power of the type system in Dπ and use a simple notion type $\mathtt{T}$ ranging over channels, $\mathtt{ch}$ and locations, $\mathtt{loc[S]}$. Location types are extended with a parameter $\mathtt{S}$ that denotes the status of the location, that is whether it is alive, $\mathtt{a}$, or else dead, $\mathtt{d}$.

As explained in the introduction, we also wish to consider the behaviour of systems under dynamic network faults. To simulate these instances, we also add to the language a construct for inducing faults, kill; even though this should not be considered part of the core language, its inclusion means that contextual equivalences will compare system behaviour in the presence of fail-stop [SS83] location failure that can occur *dynamically*, at any stage of the computation.

The second syntactic category, ranged over by N, M, *systems*, is similar to that category in Dπ. They consist of *located processes*, terms of the form $l[\![P]\!]$, which can be composed together with the parallel operator $N \mid M$ and scoped to share private names as $(\nu\, n\!:\!\mathtt{T})N$. Note that, as with local processes, scoped names always have associated types; in the case of locations, these type carry the state of the scoped location (dead or alive).

In contrast to Dπ, DπLoc uses also an additional third level of *configurations*. At this level, we have a representation of the network on which the system is running. A typical configuration takes the form

$$\Pi \triangleright N$$

where Π represents the network state. This network representation is made up of two components $\langle \mathcal{N}, \mathcal{A}\rangle$:

- $\mathcal{N}$ is a set of all the free names used in the system N of the configuration. It contains both channel and location names and thus satisfies the condition $\mathcal{N} \subseteq$ Names.
- $\mathcal{A}$ is referred to as a liveset, representing locations that are active and allow location computation to happen. Since in DπLoc we only consider location failure, $\mathcal{A}$ is a set of live location names, $\mathcal{A} \subseteq \mathbf{loc}(\mathcal{N})$, where $\mathbf{loc}(\mathcal{N})$ denotes the location names in $\mathcal{N}$.

For an arbitrary Π we use $\Pi_{\mathcal{N}}$ and $\Pi_{\mathcal{A}}$ to refer to its individual components. Throughout the chapter, we refer to a number of judgements based on the state of the network. We here give the definition for two of these judgements; the reader is referred to the Appendix for a complete listing.

$$\begin{array}{rcll} \Pi \vdash l : \textbf{alive} & \stackrel{\text{def}}{=} & l \in \Pi_{\mathcal{A}} & (l \textit{ is alive in } \Pi) \\ \Pi \vdash k \leftarrow l & \stackrel{\text{def}}{=} & l, k \in \Pi_{\mathcal{A}} & (k \textit{ accessible from } l \textit{ in } \Pi) \end{array}$$

A location l is dead in Π if it is declared ($l \in \Pi_{\mathcal{N}}$) but not in the liveset ($l \notin \Pi_{\mathcal{A}}$). Similarly, k is inccessible from l in Π if k is dead. We also use a number of operations on Π; we here define the most important one, location killing, which translates to adding a location names to the dead-set, and leave the rest to the Appendix.

$$\Pi - l \quad \stackrel{\text{def}}{=} \begin{cases} \langle \Pi_{\mathcal{N}}, \Pi_{\mathcal{A}}/\{l\} \rangle & \text{if } l \in \Pi_{\mathcal{N}} \\ \Pi & \text{otherwise} \end{cases} \qquad (\textit{killing } l \textit{ in } \Pi)$$

Notation 2.1.1. The input constructs are binders for variables, while the scoping constructs $(\nu\, n : \mathtt{T})N$ and $(\nu\, n : \mathtt{T})P$ are binders for names. We assume the usual concepts of *free* and *bound* occurrences, and the associated notation, such as α-conversion and capture avoiding substitution of names for variables. Terms with no occurrences of free variables are called *closed*, and in the sequel, we will assume that all system level terms and configurations are closed.

Throughout the report a number of abbreviations are used to improve the readability of code. We often omit occurrences of $\mathbf{0}$ in synchronous constructs like input, output and conditional constructs. Thus, $a?(X)$, $a!\langle V \rangle$, $\mathsf{if}\ n = m.P$ and $\mathsf{ping}\ l.\lceil Q \rceil$ are shorthand for $a?(X).\mathbf{0}$, $a!\langle V \rangle.\mathbf{0}$, $\mathsf{if}\ n = m.P\lceil \mathbf{0} \rceil$ and $\mathsf{ping}\ l.\mathbf{0}\lceil Q \rceil$ respectively. Similarly, the abbreviation $\mathsf{if}\, n \neq m.P$ is stands for $\mathsf{if}\, n = m.\mathbf{0}\lceil P \rceil$. Also $a?().P$ denotes an input where the binding variable does not occur in P and $a!\langle\rangle.P$ denotes an output where no value is sent. We also write $\mathsf{go}\ l_1, \ldots, l_n.P$ as an abbreviation to the nested moves $\mathsf{go}\ l_1.(\ldots(\mathsf{go}\ l_n.P))$. Finally we will also omit occurrences of types from terms, unless they are relevant to the discussion at hand. ■

2.2 The reduction semantics of DπLoc

The reduction semantics of DπLoc is defined as a binary relation between *well-formed configurations*.

Definition 2.2.1 (Well-Formed Configurations). A configuration $\Pi \triangleright N$ is said to be *well formed* if every free name occurring in N is in $\Pi_{\mathcal{N}}$. ■

The judgements of the reduction semantics are therefore of the form

$$\Pi \triangleright M \quad \longrightarrow \quad \Pi' \triangleright M'$$

where $\Pi \triangleright M$, $\Pi' \triangleright M'$ are well-formed configurations. The relation is defined to be the least one which satisfies the set of of rewriting rules in Table 2, Table 4 and Table 3. In the first

Table 2. *Local Reduction Rules for DπLoc*

Assuming $\Pi \vdash l : \textbf{alive}$

(r-comm)

$$\overline{\Pi \triangleright l[\![a!\langle V\rangle.P]\!] \mid l[\![a?(X).Q]\!] \longrightarrow \Pi \triangleright l[\![P]\!] \mid l[\![Q\{^V/_X\}]\!]}$$

(r-rep)

$$\overline{\Pi \triangleright l[\![*a?(X).P]\!] \longrightarrow \Pi \triangleright l[\![a?(X).(P \mid *a?(X).P)]\!]}$$

(r-fork)

$$\overline{\Pi \triangleright l[\![P|Q]\!] \longrightarrow \Pi \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(r-eq)

$$\overline{\Pi \triangleright l[\![\mathsf{if}\ u = u.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![P]\!]}$$

(r-neq)

$$\overline{\Pi \triangleright l[\![\mathsf{if}\ u = v.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![Q]\!]}\ u \neq v$$

batch of reduction rules (Table 2) we adapt the standard axioms for reduction in Dπ from [HR98]. The main modification is that all reductions, such as communication in the rule (r-comm) and testing for equality between identifiers, (r-eq) and (r-neq), require the location of the activity be alive in the network; there is the global requirement that $\Pi \vdash l : \textbf{alive}$. As stated earlier, we use various notation for checking the status of a network, or updating it; this will be explained informally as it is introduced, with the formal definitions relegated to the Appendix.

The rules for the novel constructs are in Table 3. Code migration is still asynchronous, but is now subject to the current state of the network: (r-go) says that if the destination location k is *accessible* from the source location l, denoted as $\Pi \vdash k \leftarrow l$, then the migration will be successful; otherwise, if k is inaccessible from l, $\Pi \nvdash k \leftarrow l$, then (r-ngo) states that the migration fails and the migrating code is lost. Similarly, the ping construct, continues as P at l if k is accessible from the current location but branches to Q at l if k is inaccessible. We note that in DπLoc, the only way for k to be inaccessible from l, is when the former is dead; in the next chapter, accessibility will also depend on the links between locations. Dynamic network faults are engendered in the obvious manner by (r-kill), and finally (r-new), allows us to export locally generated new names to the system level, as in Dπ.

The rules in Table 4 are adaptations of standard rules for the π-calculus. For instance, the first rule, (r-str), states that the reduction semantics is defined up to a *structural equivalence*, defined in the usual manner, as the least equivalence relation on systems which satisfies the set of rules and axioms in Table 5. The remaining reduction rules in Table 4 state that reduction is preserved by parallel composition and name scoping operations on configurations. But note that the rule for scoping, (r-ctxt-rest), uses an obvious notation $\Pi + n : \mathtt{T}$ for extending network representations with new names, which is formally defined in the Appendix. Note also that this rule needs to allow for the type of the scoped name to change; this is because types for locations actually carry *dynamic* state information, namely whether they are alive or dead, as explained in the following

Table 3. *Network Reduction Rules for DπLoc*

Assuming $\Pi \vdash l : \textbf{alive}$

(r-go)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Pi \triangleright k[\![P]\!]}\ \Pi \vdash k \leftarrow l$$

(r-ngo)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Pi \triangleright k[\![\mathbf{0}]\!]}\ \Pi \nvdash k \leftarrow l$$

(r-ping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![P]\!]}\ \Pi \vdash k \leftarrow l$$

(r-nping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Pi \triangleright l[\![Q]\!]}\ \Pi \nvdash k \leftarrow l$$

(r-new)
$$\frac{}{\Pi \triangleright l[\![(\nu\, n{:}\mathtt{T})P]\!] \longrightarrow \Pi \triangleright (\nu\, n{:}\mathtt{T})\, l[\![P]\!]}$$

(r-kill)
$$\frac{}{\Pi \triangleright l[\![\mathsf{kill}]\!] \longrightarrow (\Pi - l) \triangleright l[\![\mathbf{0}]\!]}$$

Table 4. *Contextual Reduction Rules for DπLoc*

(r-str)
$$\frac{\Pi \triangleright N' \equiv \Pi \triangleright N \qquad \Pi \triangleright N \longrightarrow \Pi' \triangleright M \qquad \Pi' \triangleright M \equiv \Pi' \triangleright M'}{\Pi \triangleright N' \longrightarrow \Pi' \triangleright M'}$$

(r-ctxt-rest)
$$\frac{\Pi + n : \mathtt{T} \triangleright N \longrightarrow \Pi' + n : \mathtt{U} \triangleright M}{\Pi \triangleright (\nu\, n : \mathtt{T})N \longrightarrow \Pi' \triangleright (\nu\, n : \mathtt{U})M}$$

(r-ctxt-par)
$$\frac{\Pi \triangleright N \longrightarrow \Pi' \triangleright N'}{\begin{array}{l}\Pi \triangleright N|M \longrightarrow \Pi' \triangleright N'|M \\ \Pi \triangleright M|N \longrightarrow \Pi' \triangleright M|N'\end{array}}\ \Pi \vdash M$$

example.

Example 2.2.2. Consider the following system

$$\Pi \triangleright k[\![\mathsf{go}\, l.a?(x).P]\!] \;\; | \, l[\![(\nu\, k_0 : \mathtt{loc[a]})(a!\langle k_0\rangle.Q \;|\; \mathsf{go}\, k_0.\mathsf{kill})]\!] \tag{2.1}$$

where Π is the network representation $\langle\{l, k, a\}, \emptyset\rangle$, consisting of the two *live* locations l, k. The addition of the construct go k_0. kill indicates that we wish to consider the newly created location k_0, as defective, and thus it may become faulty some time in the future.

As in Dπ, an application of (r-go), based on the fact that both k and l are alive, and (r-par-ctxt) on (2.1), yields

$$\Pi \triangleright l[\![a?(x).P]\!] \;\; | \, l[\![(\nu\, k_0 : \mathtt{loc[a]})a!\langle k_0\rangle.Q \;|\; \mathsf{go}\, k_0.\mathsf{kill}]\!]$$

which can be followed by an application of (r-fork), (r-new) (and (r-par-ctxt)) to launch a new location k_0 and get

$$\Pi \triangleright l[\![a?(x).P]\!] \;\; | \, (\nu\, k_0 : \mathtt{loc[a]})(\, l[\![a!\langle k_0\rangle.Q]\!] \;|\; l[\![\mathsf{go}\, k_0.\mathsf{kill}]\!]\,) \tag{2.2}$$

Table 5. *Structural Rules for DπLoc*

(s-comm)	$N\|M$	$\equiv$	$M\|N$	
(s-assoc)	$(N\|M)\|M'$	$\equiv$	$N\|(M\|M')$	
(s-unit)	$N\|l[\![\mathbf{0}]\!]$	$\equiv$	N	
(s-extr)	$(\nu\, n:\mathtt{T})(N\|M)$	$\equiv$	$N\|(\nu\, n:\mathtt{T})M$	$n \notin \mathbf{fn}(N)$
(s-flip)	$(\nu\, n:\mathtt{T})(\nu\, m:\mathtt{U})N$	$\equiv$	$(\nu\, m:\mathtt{U})(\nu\, n:\mathtt{T})N$	
(s-inact)	$(\nu\, n:\mathtt{T})N$	$\equiv$	N	$n \notin \mathbf{fn}(N)$

Subsequently, we can perform a communication on channel a using (r-par-comm), thereby enlarging the scope of $(\nu\, k_0:\mathtt{loc[a]})$ through the structural rule (s-extr) and obtain

$$\Pi \triangleright (\nu\, k_0:\mathtt{loc[a]})(\; l[\![P\{^{k_0}/_x\}]\!] \;\; |\; l[\![Q]\!] \;\; |\; l[\![\mathsf{go}\; k_0.\mathsf{kill}]\!]\;) \tag{2.3}$$

At this point we can analyse the novel reductions in DπLoc. In (2.3) the fault inducing process $\mathsf{go}\; k_0.\mathsf{kill}$ can move to k_0 since k_0 is described as alive by the type $\mathtt{loc[a]}$, thereby obtaining

$$\Pi \triangleright (\nu\, k_0:\mathtt{loc[a]})(\; l[\![P\{^{k_0}/_x\}]\!] \;\; |\; l[\![Q]\!] \;\; |\; k_0[\![\mathsf{kill}]\!]\;) \tag{2.4}$$

Finally, (r-kill), followed by (r-ctxt-par), can be used to kill k_0 and derive

$$(\Pi + k_0:\mathtt{loc[a]}) \triangleright l[\![P\{^{k_0}/_x\}]\!] \;\; |\; l[\![Q]\!] \;|\; k_0[\![\mathsf{kill}]\!] \quad \longrightarrow$$
$$(\Pi + k_0:\mathtt{loc[d]}) \triangleright l[\![P\{^{k_0}/_x\}]\!] \;\; |\; l[\![Q]\!] \;|\; k_0[\![\mathbf{0}]\!]$$

where the type of k_0 changes from $\mathtt{loc[a]}$ to $\mathtt{loc[d]}$, and thus, an application of (r-ctxt-rest) reduces (2.4) to

$$\Pi \triangleright (\nu\, k_0:\mathtt{loc[d]})(\; l[\![P\{^{k_0}/_x\}]\!] \;\; |\; l[\![Q]\!] \;\; |\; k_0[\![\mathbf{0}]\!]\;) \tag{2.5}$$

■

This example also serves to illustrate another important point that we shall refer to repeatedly in this report. In general, in a configuration

$$\Pi \triangleright N$$

Π denotes the network representation on which the system N is running. But there may be subsystems of N which are running on extended (internal) networks. For example in (2.2) above, the subsystem $l[\![a?(x).P]\!]$ is running with respect to the network Π, while the subsystem $l[\![a!\langle k_0\rangle.Q]\!] \;\; |\; l[\![\mathsf{go}\; k_0.\mathsf{kill}]\!]$ is running with respect to $(\Pi + k_0:\mathtt{loc[a]})$.

2.2.1 Reduction barbed congruence

In view of the reduction semantics, we can now adapt the standard approach [HR04, HMR04] to obtain a contextual equivalence for DπLoc. There are various candidates for such a contextual equivalence, namely *testing equivalence* [NH84], but we use a variation of

reduction barbed conguence, first proposed in [HY95]. We wish to compare the behaviour of systems running on the same network and thus use the following framework, borrowed from [HMR04]:

Definition 2.2.3 (Typed Relation). A *typed relation* over systems is a family of binary relations between systems, $\mathcal{R}$, indexed by network representations. We write $\Pi \models M \mathrel{\mathcal{R}} N$ to mean that systems M and N are related by $\mathcal{R}$ at index Π, that is $M \mathrel{\mathcal{R}_\Pi} N$, and moreover $\Pi \triangleright M$ and $\Pi \triangleright N$ are valid configurations. ■

The definition of our equivalence hinges on what it means for a typed relation to be *contextual*, which must take into account the presence of the network. Our definition has two requirements:

- systems running on the network Π must be considered equivalent by all observers also running on Π
- systems must remain equivalent when the network is extended by new locations.

First let us define what kinds of observing systems are allowed to run on a given network.

Definition 2.2.4 (Observers). The intuition of *valid* observer system O in a distributed setting Π, denoted as $\Pi \vdash_{\mathsf{obs}} O$, is that O originates from some *live fresh* location k_0, migrates to any location in $\mathbf{loc}(\Pi_{\mathcal{N}})$ to interact with (observe) processes there and then returns back to the originating fresh location k_0 to compare its observations with other observers. For convenience, we do not mention fresh locations k_0 and place observing code immediately at locations in $\mathbf{loc}(\Pi_{\mathcal{N}})$. We note that, according to the definition of the reduction rule (r-ngo), observing code can never reach dead locations and we therefore have to encode this in our definition of $\Pi \vdash_{\mathsf{obs}} O$. For convenience, we also disallow observer to be located at scoped *dead* loactions, that is observers of the form $(\nu\, l\!:\!\mathtt{loc}[\mathtt{d},)]N$; to denote all other forms of allowed types for scoped observer names we use the notation $\Pi \vdash_{\mathsf{obs}} \mathtt{T}$, which is defined in the Appendix. Thus, $\Pi \vdash_{\mathsf{obs}} O$ is recursively defined as:-

- $\Pi \vdash_{\mathsf{obs}} l[\![P]\!]$ if $\mathbf{fn}(P) \subseteq \Pi_{\mathcal{N}}$ and $\Pi \vdash l : \mathbf{alive}$
- $\Pi \vdash_{\mathsf{obs}} (\nu\, n\!:\!\mathtt{T})N$ if $\Pi \vdash_{\mathsf{obs}} \mathtt{T}$ and $(\Pi + n\!:\!\mathtt{T}) \vdash_{\mathsf{obs}} N$
- $\Pi \vdash_{\mathsf{obs}} M \,|\, N$ if $\Pi \vdash_{\mathsf{obs}} M$ and $\Pi \vdash_{\mathsf{obs}} N$ ■

Definition 2.2.4, defining allowed observer systems, determines the definition of our sepcific definition of *contextuality* given below.

Definition 2.2.5 (Contextual typed relations). A typed relation $\mathcal{R}$ over configurations is *contextual* if:

(Parallel Systems)

- $\Pi \models M \mathrel{\mathcal{R}} N$ and $\Pi \vdash_{\mathsf{obs}} O$ implies
  - $\Pi \models M|O \mathrel{\mathcal{R}} N|O$
  - $\Pi \models O|M \mathrel{\mathcal{R}} O|N$

(Network Extensions)

- $\Pi \models M \mathrel{\mathcal{R}} N$ and $\Pi \vdash_{\mathsf{obs}} \mathtt{T}$, n fresh implies $\Pi + n\!:\!\mathtt{T} \models M \mathrel{\mathcal{R}} N$

■

Definition 2.2.6 (Reduction barbed congruence). First we define the adaptation of the other standard relations required to define *reduction barbed congruence*.

- $\Pi \triangleright N \Downarrow_{a@l}$ denotes an *observable barb* exhibited by the configuration $\Pi \triangleright N$, on channel a at location l. Formally, it means that $\Pi \triangleright N \longrightarrow^{*} \Pi' \triangleright N'$ for some $\Pi' \triangleright N'$ such that $N' \equiv M|l[\![a!\langle V\rangle.Q]\!]$ and $\Pi \vdash l : \textbf{alive}$. Then, we say a typed relation $\mathcal{R}$ over configurations is *barb preserving* whenever $\Pi \models N \;\; \mathcal{R} \;\; M$ and $\Pi \triangleright N \Downarrow_{a@l}$ implies $\Pi \triangleright M \Downarrow_{a@l}$.
- A typed relation $\mathcal{R}$ over configurations is *reduction closed* whenever $\Pi \models N \;\mathcal{R}\; M$ and $\Pi \triangleright N \longrightarrow \Pi' \triangleright N'$ implies $\Pi \triangleright M \longrightarrow^{*} \Pi' \triangleright M'$ for some $\Pi' \triangleright M'$ such that $\Pi' \models N' \;\mathcal{R}\; M'$.

Then $\cong$, called *reduction barbed congruence*, is the largest symmetric typed relation over configurations which is:

- barb preserving
- reduction closed
- contextual ■

We leave the reader to check that pointwise $\cong$ is an equivalence relation.

Example 2.2.7. Consider the systems onePkt and twoPkt defined as:

$$\begin{aligned} \mathsf{onePkt} &\Leftarrow l[\![\mathsf{go}\, k.(a!\langle\rangle|b!\langle\rangle)]\!] \\ \mathsf{twoPkt} &\Leftarrow l[\![\mathsf{go}\, k.a!\langle\rangle]\!] \,|\, l[\![\mathsf{go}\, k.b!\langle\rangle]\!] \end{aligned}$$

They represent two different strategies for sending the messages $a!\langle\rangle|b!\langle\rangle$ from l to k. The first system, onePkt, transfers the two messages as one unit (one packet), whereas the second system, twoPkt, uses a distinct packet for every message. In a calculus with no network failure, it would be hard to distinguish between these two systems.

The two configurations are however not reduction barbed congruent in our calculus when run over the network $\Pi_{lk} = \langle\{l,k,a,b\};\emptyset\rangle$, in which l, k are alive. This is formally stated as

$$\Pi_{lk} \models \mathsf{onePkt} \not\cong \mathsf{twoPkt}$$

and the reason why they are not is because they can exhibit different behaviour when l is subject to failure during the transfer of the packets. Formally, we can examine the behaviour of systems under this situation by considering their behaviour in the context

$$C[-] \;=\; [-]\,|\,l[\![\mathsf{kill}]\!]$$

By Definition 2.2.6, if we assume that $\Pi_{lk} \models \mathsf{onePkt} \cong \mathsf{twoPkt}$, then *contextuality* of $\cong$ would imply

$$\Pi_{lk} \triangleright \mathsf{onePkt}\,|\,l[\![\mathsf{kill}]\!] \;\cong \Pi_{lk} \triangleright \mathsf{twoPkt}\,|\,l[\![\mathsf{kill}]\!]$$

But we can show directly that the latter cannot be true, thereby contradicting our assumption. For example, using the reduction rules of Tables 2, 4 and 3 we can derive the following sequence of reductions for $\Pi_{lk} \triangleright \mathsf{twoPkt} \,|\, l[\![\mathsf{kill}]\!]$:

$$\begin{aligned}
\Pi_{lk} \triangleright l[\![\mathsf{go}\ k.a!\langle\rangle]\!] \mid l[\![\mathsf{go}\ k.b!\langle\rangle]\!] \mid l[\![\mathsf{kill}]\!] &\longrightarrow \Pi_{lk} \triangleright k[\![a!\langle\rangle]\!] \mid l[\![\mathsf{go}\ k.b!\langle\rangle]\!] \mid l[\![\mathsf{kill}]\!] \\
&\longrightarrow \Pi_{k} \triangleright k[\![a!\langle\rangle]\!] \mid l[\![\mathsf{go}\ k.b!\langle\rangle]\!] \\
&\longrightarrow \Pi_{k} \triangleright k[\![a!\langle\rangle]\!]
\end{aligned}$$

where Π_k is the network representation in which l is dead, that is $\langle\{l,k,a,b\};\{l\}\rangle$ We also note that

$$\Pi_k \triangleright k[\![a!\langle\rangle]\!] \Downarrow_{a@k}$$
$$\Pi_k \triangleright k[\![a!\langle\rangle]\!] \not\Downarrow_{b@k}$$

However the left hand side, $\Pi_{lk} \triangleright \mathsf{onePkt} \,|\, l[\![\mathsf{kill}]\!]$ can never reduce to a configuration with such barbs. Formally, there is no configuration $\Pi \triangleright N$ such that

$$\Pi_{lk} \triangleright l[\![\mathsf{go}\ k.(a!\langle\rangle|b!\langle\rangle)]\!] \,|\, l[\![\mathsf{kill}]\!] \longrightarrow^{*} \Pi \triangleright N$$

where $\Pi \triangleright N \Downarrow_{a@k}$ and $\Pi \triangleright N \not\Downarrow_{b@k}$. ■

Example 2.2.8. Consider the two systems:

$$\begin{aligned}
\mathsf{nonDet1} &\Leftarrow (\nu\, k\!:\!\texttt{loc[a]})\ k[\![\mathsf{kill}]\!] \,|\, k[\![\mathsf{go}\ l.a!\langle\rangle]\!] \\
\mathsf{nonDet2} &\Leftarrow (\nu\, b\!:\!\mathsf{ch})\ l[\![b!\langle\rangle]\!] \,|\, l[\![b?()]\!] \,|\, l[\![b?().a!\langle\rangle]\!]
\end{aligned}$$

Both systems exhibit a barb $a@l$ depending on different forms of *non-deterministic internal choices*; the internal choice used by nonDet1 is based on a scoped location k that may fail while the internal choice used by nonDet2 is based on two inputs competing for a single scoped output on channel b.

It turns out that these two systems are observationally equivalent when run over the simple network $\Pi_l = \langle\{l,a\},\emptyset\rangle$, formally stated as

$$\Pi_l \models \mathsf{nonDet1} \cong \mathsf{nonDet2}$$

Nevertheless, Definition 2.2.6 exhibits a major limitation at this point, because it makes it quite hard to prove such an equivalence. Such a complication arises from the fact that the definitions of reduction barbed congruence requires us to reason about the behaviour of the two configurations under all possible contexts, which are infinitely many. ■

Example 2.2.9. Here we consider three implementations of a simple (abstract) server, executing on a network $\Pi = \langle\{l,k_1,k_2,serv,ret\};\emptyset\rangle$ where three locations l, k_1 and k_2 are alive. The first is the most straightforward:

$$\mathsf{server} \Leftarrow (\nu\, data\!:\!\mathsf{ch})(l[\![req?(x,y).data!\langle x,y\rangle]\!] \mid l[\![data?(x,y).y!\langle f(x)\rangle]\!])$$

It simply takes in a request at the port req consisting of an argument, x, and a return channel on which to return the answer of the request, y. The server proceeds by forwarding the

two parameters, x and y to an internal database, denoted by the scoped channel *data*; intuitively, the database looks up the mapping of the value x using some unspecified function $f()$ and returns the answer, $f(x)$, back onport y. The key aspect of this server is that all the processing is performed *locally* at location l. A typical client for such a server would have the following form, sending the name l as the value to be processed and *ret* as the return channel:

$$\mathsf{client} \quad \Leftarrow \quad l[\![req!\langle l, ret\rangle]\!]$$

By contrast, the next two server implementations introduce a degree of *distribution*, by processing the request across a number of locations:

$$\mathsf{srvDis} \quad \Leftarrow \quad (\nu\, data\!:\!\mathsf{ch})\begin{pmatrix} l[\![req?(x,y).\mathsf{go}\; k_1.data!\langle x,y\rangle]\!] \\ \mid k_1[\![data?(x,y).\mathsf{go}\; l.y!\langle f(x)\rangle]\!] \end{pmatrix}$$

$$\mathsf{srv2Rt} \quad \Leftarrow \quad (\nu\, data\!:\!\mathsf{ch})\begin{pmatrix} l\left[\!\!\left[req?(x,y).(\nu\, sync\!:\!\mathsf{ch})\begin{pmatrix} \mathsf{go}\; k_1.data!\langle x, sync\rangle \\ \mid \mathsf{go}\; k_2, k_1.\, data!\langle x, sync\rangle \\ \mid synch?(x).y!\langle x\rangle \end{pmatrix} \right]\!\!\right] \\ \mid k_1 \left[\!\!\left[data?(x,y).\begin{pmatrix} \mathsf{go}\; l.\, y!\langle f(x)\rangle \\ \mathsf{go}\; k_2, l.\, y!\langle f(x)\rangle \end{pmatrix} \right]\!\!\right] \end{pmatrix}$$

Both servers, srvDis and srv2Rt, distributed the internal database *remotely* at location k_1. Server srvDis thus receives a client request at l, migrates *directly* to k_1 and queries the database; the database then returns to l and reports back the processed value, $f(x)$, on the requested return channel y. The other server, srv2Rt, accepts a client request at l, but attempts to access the unique remote database located at k_1 through *two different routes*, one *directly* from l to k_1 and the other *indirectly* from l through the intermediate node k_2 and then finally to k_1 where the database resides; similarly, the internal database of srv2Rt returns the answer $f(x)$ on y along these two routes. In a scenario where no fault occurs to k_1 and k_2, srv2Rt will receive two answers back at l. To solve this, the original requests are sent with a scoped return channel *sync*; a process waiting for answers on this channel at location l chooses non-deterministically between any two answers received and relays the answer on the original channel y.

We leave the reader to check that the local server, server and remote implementations, srvDis and srv2Rt, are different, that is:

$$\Pi \models \mathsf{server} \not\cong \mathsf{srvDis} \quad \text{and} \quad \Pi \models \mathsf{server} \not\cong \mathsf{srv2Rt}$$

because of their behaviour in the context

$$C_2[-] \quad = \quad [-] \,|\, k_1[\![\mathsf{kill}]\!]$$

However, it turns out that the two remote server implementations are reduction barbed congruent in DπLoc:

$$\Pi \models \mathsf{srvDis} \cong \mathsf{srv2Rt}$$

Again, Definition 2.2.6 makes it hard to prove this statement because it uses quantification over all possible contexts. ■

Table 6. *Operational Rules(1) for DπLoc*

Assuming $\Pi \vdash l : \textbf{alive}$

(l-out)
$$\frac{}{\Pi \triangleright l[\![a!\langle V\rangle.P]\!] \xrightarrow{l:a!\langle V\rangle} \Pi \triangleright l[\![P]\!]}$$

(l-in)
$$\frac{}{\Pi \triangleright l[\![a?(X).P]\!] \xrightarrow{l:a?(V)} \Pi \triangleright l[\![P\{^{V}/_{X}\}]\!]}\ V \subseteq \Pi_N$$

(l-in-rep)
$$\frac{}{\Pi \triangleright l[\![*a?(X).P]\!] \xrightarrow{\tau} \Pi \triangleright l[\![a?(X).(P \mid *a?(Y).P\{^{Y}/_{X}\})]\!]}$$

(l-fork)
$$\frac{}{\Pi \triangleright l[\![P \mid Q]\!] \xrightarrow{\tau} \Pi \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(l-eq)
$$\frac{}{\Pi \triangleright l[\![\mathsf{if}\ u = u.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![P]\!]}$$

(l-neq)
$$\frac{}{\Pi \triangleright l[\![\mathsf{if}\ u = v.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![Q]\!]}\ u \neq v$$

Due to the problems associated with Definition 2.2.6, we need an inductive definition of behavioural equivalence that is easier to prove but still consistent with reduction barbed congruence. In the remainder of this section we define a *bisimulation equivalence* which allows us to relate DπLoc configurations in a tractable manner. It will turn out that this bisimulation equivalence coincides with reduction barbed congruence.

2.3 A bisimulation equivalence for DπLoc

We start by defining the labelled transition system on which we base our definitions of bisimulation equivalence.

Definition 2.3.1 (A labelled transition system for DπLoc). This consists of a collection of actions $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$, where μ takes one of the forms:

- τ, representing internal action
- $(\tilde{n} : \tilde{\mathrm{T}})l : a?(V)$, representing the input of the value V along the channel a, located at l. Here $\tilde{n} : \tilde{\mathrm{T}}$ denotes the fresh names $\tilde{n}$ and their respective state information $\tilde{\mathrm{T}}$, introduced by an observer (context) as part of this action.
- $(\tilde{n} : \tilde{\mathrm{T}})l : a!\langle V\rangle$, the output of the value V along the channel a, located at l. Here $\tilde{n} : \tilde{\mathrm{T}}$ represented the names $\tilde{n}$ which are exported to an observer (context) as part of this action, together with their associated new state information $\tilde{\mathrm{T}}$.
- $\mathsf{kill} : l$, representing the killing of location l by an observer (context). ■

The transitions in the lts for DπLoc are defined as the least relations satisfying the axioms and rules in Tables 6, 8 and 7. Table 6 contains standard operational rules inherited from distributed π-calculi such as Dπ; note, however, that actions can only occur at live locations. The rules in Table 7 are also adaptations of the standard rules for actions-in-context from [HR04] together with the rule (l-par-comm), for local communication. Here,

Table 7. *Operational Rules(2) for DπLoc*

(l-open)

$$\frac{\Pi + n:\mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Pi' \triangleright N'}{\Pi \triangleright (\nu\, n:\mathtt{T})N \xrightarrow{(n:\mathtt{T},\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Pi' \triangleright N'}\ l,a \neq n \in V$$

(l-weak)

$$\frac{\Pi + n:\mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Pi' \triangleright N'}{\Pi \triangleright N \xrightarrow{(n:\mathtt{T},\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Pi' \triangleright N'}\ l,a \neq n \in V$$

(l-rest)

$$\frac{\Pi + n:\mathtt{T} \triangleright N \xrightarrow{\mu} \Pi' + n:\mathtt{U} \triangleright N'}{\Pi \triangleright (\nu\, n:\mathtt{T})N \xrightarrow{\mu} \Pi' \triangleright (\nu\, n:\mathtt{U})N'}\ n \notin \mathbf{fn}(\mu)$$

(l-par-ctxt)

$$\frac{\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'}{\begin{array}{c}\Pi \triangleright N|M \xrightarrow{\mu} \Pi' \triangleright N'|M \\ \Pi \triangleright M|N \xrightarrow{\mu} \Pi' \triangleright M|N'\end{array}}\ \Pi \vdash M$$

(l-par-comm)

$$\frac{\Pi \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Pi' \triangleright N' \qquad \Pi \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Pi'' \triangleright M'}{\begin{array}{c}\Pi \triangleright N|M \xrightarrow{\tau} \Pi \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})(N'|M') \\ \Pi \triangleright M|N \xrightarrow{\tau} \Pi \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})(M'|N')\end{array}}$$

we highlight the rule (l-weak), dealing with the learning of the existence of new location names and their state as a result of an input from the context; this rule was adopted from a variant used already in [HR04, HMR04]. Note also the general form of (l-rest), where the type of n may change from $\mathtt{T}$ to $\mathtt{U}$; this phenomena is inherited directly from (r-ctxt-rest) of Table 4 in the reduction semantics and explained in Example 2.2.9.

The rules dealing with the new constructs of DπLoc, are contained in Table 8, most of which are inherited from the reduction semantics. The only new one is (l-halt), where the action kill : l represents a failure induced by an observer. This is in contrast with the rule (l-kill), where l is killed by the system itself and the associated action is τ[1].

Using the lts of actions we can now define, in the standard manner, *weak bisimulation equivalence* over configurations. Our definition uses the standard notation for weak actions, namely $\stackrel{\mu}{\Longrightarrow}$ denotes $\Longrightarrow\stackrel{\mu}{\longrightarrow}\Longrightarrow$, and $\stackrel{\widehat{\mu}}{\Longrightarrow}$ denotes

- $\stackrel{\tau}{\longrightarrow}^*$ if $\mu = \tau$
- $\stackrel{\mu}{\Longrightarrow}$ otherwise.

Definition 2.3.2 (Weak bisimulation equivalence). This is denoted as $\approx$, and is defined to be the largest typed relation over configurations such that if $\Pi \models M \approx N$ then

- $\Pi \triangleright M \xrightarrow{\mu} \Pi' \triangleright M'$ implies $\Pi \triangleright N \stackrel{\hat{\mu}}{\Longrightarrow} \Pi' \triangleright N'$ such that $\Pi' \models M' \approx N'$
- $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$ implies $\Pi \triangleright M \stackrel{\hat{\mu}}{\Longrightarrow} \Pi' \triangleright M'$ such that $\Pi' \models M' \approx N'$ ■

[1]The details here regarding labels dealing with failure injection differ slightly than those in [RH01]

Table 8. *Operational Rules(3) for DπLoc*

Assuming $\Pi \vdash l : \textbf{alive}$

(l-kill)
$$\frac{}{\Pi \triangleright l[\![\mathsf{kill}]\!] \xrightarrow{\tau} (\Pi - l) \triangleright l[\![\mathbf{0}]\!]}$$

(l-halt)
$$\frac{}{\Pi \triangleright N \xrightarrow{\mathsf{kill}:l} (\Pi - l) \triangleright N}$$

(l-new)
$$\frac{}{\Pi \triangleright l[\![(\nu\, n : \mathtt{T})P]\!] \xrightarrow{\tau} \Pi \triangleright (\nu\, n : \mathtt{T}) l[\![P]\!]}$$

(r-go)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\, k.P]\!] \xrightarrow{\tau} \Pi \triangleright k[\![P]\!]}\ \Pi \vdash k \leftarrow l$$

(r-ngo)
$$\frac{}{\Pi \triangleright l[\![\mathsf{go}\, k.P]\!] \xrightarrow{\tau} \Pi \triangleright k[\![\mathbf{0}]\!]}\ \Pi \nvdash k \leftarrow l$$

(r-ping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\, k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![P]\!]}\ \Pi \vdash k \leftarrow l$$

(r-nping)
$$\frac{}{\Pi \triangleright l[\![\mathsf{ping}\, k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Pi \triangleright l[\![Q]\!]}\ \Pi \nvdash k \leftarrow l$$

Equipped with our bisimulation definitions, we revisit some equivalence examples introduced in § 2.2.1 and show that they can be tractably proved to be equivalent. But before, we prove a useful result that allows us to give bisimulations *up-to* structural equivalence.

Proposition 2.3.3 (Structural equivalence and bisimulation). Let us define structural equivalence over configurations in the obvious way, overloading the symbol $\equiv$, that is:

$$\Pi \triangleright M \equiv \Pi \triangleright N \;\text{ iff }\; M \equiv N \text{ and } \Pi \vdash M, N$$

Similar to any typed relation so far, we abbreviate $\Pi \triangleright M \equiv \Pi \triangleright N$ to $\Pi \models M \equiv N$. We now can state that structural equivalence over configurations is a bisimulation relation. Stated otherwise, $\equiv \subseteq \approx$

Proof. We proceed by defining the $\mathcal{R}$ as:

$$\mathcal{R} \;=\; \{\Pi \models M \,\mathcal{R}\, N \mid \Pi \models M \equiv N\}$$

It is clear that $\mathcal{R}$ is a typed relation; we only have to show that $\mathcal{R}$ it is a bisimulation. The proof proceeds by induction on the structure of $\Pi \triangleright M$ and $\Pi \triangleright N$. □

Example 2.3.4. We recall that in Example 2.2.8, we claimed that $\Pi_l \triangleright \mathsf{nonDet1}$ was equivalent to $\Pi_l \triangleright \mathsf{nonDet2}$. We here show that they are *bisimilar*, by giving the relation $\mathcal{R}$ defined as:

$$\mathcal{R} \;=\; \left\{ \begin{array}{lcl} \langle \Pi_l \triangleright M & , & \Pi_l \triangleright N \rangle \\ \langle \Pi_l - l \triangleright M & , & \Pi_l - l \triangleright N \rangle \end{array} \;\middle|\; \langle M, N \rangle \in \mathcal{R}_{sys} \right\}$$

where:

$$\mathcal{R}_{sys} = \left\{ \begin{array}{lcl} \langle \mathsf{nonDet1} & , & \mathsf{nonDet2} \rangle \\ \langle (\nu\, k\!:\!\mathtt{loc[a]})\, k[\![\mathsf{kill}]\!] \,|\, l[\![a!\langle\rangle]\!] & , & (\nu\, b\!:\!\mathsf{ch})\, l[\![b?()]\!] \,|\, l[\![a!\langle\rangle]\!] \rangle \\ \langle (\nu\, k\!:\!\mathtt{loc[d]})\, l[\![a!\langle\rangle]\!] & , & (\nu\, b\!:\!\mathsf{ch})\, l[\![b?()]\!] \,|\, l[\![a!\langle\rangle]\!] \rangle \\ \langle (\nu\, k\!:\!\mathtt{loc[d]})\, k[\![\mathsf{go}\, l.a!\langle\rangle]\!] & , & (\nu\, b\!:\!\mathsf{ch})\, l[\![b?().a!\langle\rangle]\!] \rangle \\ \langle (\nu\, k\!:\!\mathtt{loc[a]})\, k[\![\mathsf{kill}]\!] & , & (\nu\, b\!:\!\mathsf{ch})\, l[\![b?()]\!] \rangle \\ \langle (\nu\, k\!:\!\mathtt{loc[d]})\, k[\![\mathbf{0}]\!] & , & (\nu\, b\!:\!\mathsf{ch})\, l[\![b?()]\!] \rangle \end{array} \right\}$$

Relation $\mathcal{R}$ is closed over all possible actions. There are two key actions in this bisimulation, that particular to our calculus with failures.

- The first action is the τ-action caused by the process $k[\![\mathsf{kill}]\!]$ when it kills the scoped location k in nonDet1. This can non-deterministically happen at any point and affects the ability of nonDet1 producing an output on a at l.
  - If the output process $a!\langle\rangle$ is still at k when it is killed, then killing k prohibits nonDet1 from producing an output on a at l. nonDet2 matches this action in $\mathcal{R}$ by a strong τ-action where $l[\![b!\langle\rangle.]\!]$ reacts with $l[\![b?()]\!]$; since b is scoped, the other located process $l[\![b?().a!\langle\rangle]\!]$ is blocked forever and can never produce an output on a at l.
  - If the output process $a!\langle\rangle$ is at l, then the τ-action killing k is matched by the empty move.
- If $k[\![\mathsf{go}\, l.a!\langle\rangle]\!]$ silently transitions to $l[\![a!\langle\rangle]\!]$ in nonDet1, then the production an output action on a at l by nonDet1 is *independent* of the state of k. This τ-action is matched by the internal communication of $l[\![b!\langle\rangle.]\!]$ with $l[\![b?().a!\langle\rangle]\!]$, which releases $a!\langle\rangle$ in nonDet2. ■

Of course we need to justify the use of bisimulations to relate systems. This is provided by the following result:

Theorem 2.3.5 (Soundness and Completeness for DπLoc). In DπLoc, $\Pi \models N \approx M$ if and only if $\Pi \models N \cong M$. ■

We omit the full proof here but outline the main propositions and lemmas required to prove Theorem 2.3.5. The full proof can be derived from the full proof given for the more complex characterisation result of Theorem 3.5.10, given in Chapter 3.

The proof of Theorem 2.3.5 is split in two phases: *soundness* and *completeness*. Soundness means that bisimilarity implies reduction barbed congruence, that is

$$\Pi \models M \approx N \quad \text{implies} \quad \Pi \models M \cong N$$

For this proof we need to show that $\approx$ satisfies the defining conditions of reduction barbed congruence, namely barb preservation, reduction closure and contextuality. The latter property, contextuality, turns out to be the hardest to prove. We thus prove two lemmas, Composition and Decomposition, describing how a transition can be composed and decomposed.

Lemma 2.3.6 (Composition).

- Suppose $\Pi \triangleright M \xrightarrow{\mu} \Pi' \triangleright M'$. If $\Pi \vdash N$ for arbitrary system N, then $\Pi \triangleright M|N \xrightarrow{\mu} \Pi' \triangleright M'|N$ and $\Pi \triangleright N|M \xrightarrow{\mu} \Pi \triangleright N|M$.
- Suppose $\Pi \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Pi' \triangleright M'$ and $\Pi \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Pi'' \triangleright N'$. Then
  - $\Pi \triangleright M|N \xrightarrow{\tau} \Pi \triangleright (\nu\, \tilde{n}:\tilde{\mathrm{T}})M'|N'$
  - $\Pi \triangleright N|M \xrightarrow{\tau} \Pi \triangleright (\nu\, \tilde{n}:\tilde{\mathrm{T}})N'|M'$

Proof. (Outline) The proof proceeds by extracting the necessary structure of the systems M, N and the network Π to be able to re-compose them using rules such as (l-par-ctxt), (l-par-comm) and (l-rest). ☐

Lemma 2.3.7 (Decomposition). Suppose $\Pi \triangleright M|N \xrightarrow{\mu} \Pi' \triangleright M'$. Then, one of the following conditions hold:

1. M' is $M''|N$, where $\Pi \triangleright M \xrightarrow{\mu} \Pi' \triangleright M''$.
2. M' is $M|N'$ and $\Pi \triangleright N \xrightarrow{\mu} \Pi' \triangleright N'$.
3. M' is $(\nu\, \tilde{n}:\tilde{\mathrm{T}})M''|N'$, μ is τ, $\Pi' = \Pi$ and either
   - $\Pi \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Pi'' \triangleright M''$ and $\Pi \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Pi''' \triangleright N'$
   - $\Pi \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Pi'' \triangleright M''$ and $\Pi \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Pi''' \triangleright N'$

Proof. (Outline) The proof proceeds by induction on the derivation of $\Pi \triangleright M|N \xrightarrow{\mu} \Pi' \triangleright M'$. ☐

Proposition 2.3.8 (Contextuality of Behavioural Equivalence). If two configurations are bisimilar, they are also bisimilar under any context. Stated otherwise, $\Pi \models N \approx M$ implies that for $\Pi \vdash_{\mathsf{obs}} O, \mathtt{T}$ and n fresh in Π we have:

- $\Pi \models N|O \approx M|O$ and $\Pi \models O|N \approx O|M$
- $\Pi + n:\mathtt{T} \models N \approx M$

Proof. (Outline) The proof progresses by the inductive definition a relation $\mathcal{R}$ as the largest typed relation over configurations satisfying:

$$\mathcal{R} = \left\{ \begin{array}{ll}
\langle \Pi \triangleright M_1,\ \Pi \triangleright M_2 \rangle & \mid \Pi \triangleright M_1 \approx \Pi \triangleright M_2 \\[2ex]
\begin{array}{l} \langle \Pi \triangleright M_1|O,\ \Pi \triangleright M_2|O \rangle \\ \langle \Pi \triangleright O|M_1,\ \Pi \triangleright O|M_2 \rangle \end{array} & \left| \ \Pi \triangleright M_1 \mathcal{R} \Pi \triangleright M_2 \right. \\[2ex]
\langle \Pi + n:\mathtt{T} \triangleright M_1,\ \Pi + n:\mathtt{T} \triangleright M_2 \rangle & \left| \begin{array}{l} \Pi_1 \triangleright M_1 \,\mathcal{R}\, \Pi_2 \triangleright M_2, \\ \Pi \vdash \mathtt{T} \text{ and } n \text{ is fresh} \end{array} \right. \\[2ex]
\langle \Pi \triangleright (\nu\, n:\mathtt{T})M_1,\ \Pi \triangleright (\nu\, n:\mathtt{T})M_2 \rangle & \mid \Pi + n:\mathtt{T} \triangleright M_1 \,\mathcal{R}\, \Pi + n:\mathtt{T} \triangleright M_2
\end{array} \right\}$$

and showing that $\mathcal{R} \subseteq \approx$. The last clause is required to guarantee the closure of $\mathcal{R}$ with respect to all possible tranitions, including scope extrusion of names. Contextuality of $\approx$ would then be a special case of the above relation, without the final clause, since $\approx$ is the biggest possible relation. □

The second part of the proof of Theorem 2.3.5 is completeness, represented as

$$\Pi \models M \cong N \quad \text{implies} \quad \Pi \models M \approx N$$

meaning that for every pair that are reduction barbed congruent we can give a bisimulation that justifies this. This proof relies on the notion of *definability*, that is for every action there is an observer which can completely characterise the effect of that action.

Proposition 2.3.9 (Definability). Assume that for an arbitrary network representation Π, the network Π_+ denotes:

$$\Pi_+ = \Pi + k_0 : \mathtt{loc[a]}, \textsc{succ} : \mathtt{ch}, \textsc{fail} : \mathtt{ch}$$

where k_0, $\textsc{succ}$ and $\textsc{fail}$ are fresh to $\Pi_{\mathcal{N}}$. Thus, for every external action μ and network representation Π, every non-empty finite set of names Nm where $\Pi_{\mathcal{N}} \subseteq Nm$, every fresh pair of channel names $\textsc{succ}$, $\textsc{fail} \notin Nm$, and every fresh location name $k_0 \notin Nm$, there exists a system $T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$ with the property that $\Pi_+ \vdash_{\mathsf{obs}} T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$, such that:

1. $\Pi \triangleright N \xrightarrow{\mu} \Pi' + \mathbf{bn}(\mu) \triangleright N'$ implies
   $\Pi_+ \triangleright N \,|\, T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Pi'_+ \triangleright (\nu\, \mathbf{bn}(\mu))\, N' \mid k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$
2. $\Pi_+ \triangleright N \,|\, T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Pi'_+ \triangleright N'$, where $\Pi'_+ \triangleright N' \Downarrow_{\textsc{succ}@k_0}$, $\Pi'_+ \triangleright N' \not\Downarrow_{\textsc{fail}@k_0}$
   implies that
   $N' \equiv (\nu\, \mathbf{bn}(\mu)) N'' | k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$ for some N'' such that $\Pi \triangleright N \stackrel{\mu}{\Longrightarrow} \Pi' + \mathbf{bn}(\mu) \triangleright N''$.

Proof. (Outline) In our case, we only need to prove definability for input/output actions, which has already been done for a more complex setting in [HMR04], and for the kill action $\mathsf{kill} : l$ which turns out to be definable using the context

$$l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle \rceil]\!]$$

□

2.4 Summary

In this Chapter we presented a distributed π-calculus that described location failure: we defined its reduction semantics, a reduction barbed congruence for systems running over the same network representation and a bisimulation that coincides with this congruence. While the calculus is very similar to [AP94, Ama97], the results obtained are an extension to those obtained in [RH01]. We conjuncture that the our choice of barbs, used in our definition of reduction barbed congruence, can be simplified to a more concise form such as $\Pi \triangleright M \Downarrow_l$ denoting an output on *some* channel at live location l (in the spirit of [DGP05])

and still retain the same reduction barbed conguence. The intuition for this is that our contexts may extended the network to fresh locations and we can thus generate a barb on some channel at a fresh location "al" for every former barb $\Pi \triangleright M \Downarrow_{a@l}$. We also conjuncture that the lts given is also appropriate (bar some minor standard adjustments) for defining a "may testing" semantics for our language that is typically contextual, barb preserving but not necessarily reduction closed [BNP99, BNP02]. The reason for believing this is that our lts already exhibits notions of co-action such as bound inputs and outputs. Most importantly however, in this Chapter we lay the foundations for the framework used in the following two chapters.

Chapter 3

Dπ with Location and Link Failure

In this section we extend the network representation to describe the state of physical links between sites. As explained in the Introduction, in such a setting we can then represent *link failures*, resulting from faults in links between locations. Moreover, the liveness of such links affects in turn the semantics of ping, the construct used to detect faults.

The core language remains the same, although we need to add a new construct to induce link faults. With this extended notion of a network we redo the previous section, obtaining similar results; however the development is considerably more complicated.

3.1 DπF syntax

The syntax of DπF is a minor extension to that of DπLoc; Table 9 highlights the novelties. The main one is that new types are required for locations. Now when a new location is declared, in addition to its live/dead *status*, we have to also describe the live *connections* to other locations. Thus, in DπF, a location type is denoted as $\mathtt{loc}[\mathtt{S},\mathtt{C}]$, where the first element $\mathtt{S}$ is inherited from Chapter 2, and the second element $\mathtt{C}$ is a set of locations $\{l_1,\ldots,l_n\}$. If a new k location is declared at this type, then it is intended to be linked in the underlying network with each of the locations l_i, although there will be complications; see Example 3.2.1. Another modificiation to the syntax is the addition of the process construct $\mathsf{break}\ l$, which breaks a live connection between the location hosting the process and location l. This construct is symetric in the sense that a link between l and k can be broken from either location; stated otherwise, $l[\![\mathsf{break}\ k]\!]$ induces the same fault as $k[\![\mathsf{break}\ l]\!]$. Contextual equivalences then take into account the effect of link faults on system behaviour, in the same manner as the presence of kill takes node faults into account.

We also change the network representation for DπF. In a setting where not every node is interconnected, the network representation needs also to represent which nodes are connected apart from their current alive/dead status.

Definition 3.1.1 (Network representation). First let us introduce some notation to represent the *links* in a network. A binary relation $\mathcal{L}$ over locations is called a *link set* if it

Table 9. *Syntax of DπF*

Types

`T,U,W`	::=	`ch` | `loc[S,C]`	`S,R`	::=	`a` | `d`
			`C,D`	::=	$\{u_1, \ldots, u_n\}$

Processes

P, Q ::= $\ldots$ | break l

Systems

N, M ::= $\ldots$

is:

- symmetric, that is, $\langle l, k\rangle \in \mathcal{L}$ implies $\langle k, l\rangle$ is also in $\mathcal{L}$
- reflexive, that is, $\langle l, k\rangle \in \mathcal{L}$ implies $\langle l, l\rangle$ and $\langle k, k\rangle$ are also in $\mathcal{L}$.

The latter property allows the smooth handling of the degenerate case of a process moving from a site l to l itself. Also for any linkset $\mathcal{L}$ we let $\mathbf{dom}(\mathcal{L})$ denote its domain; that is the collection of locations l, such that $\langle l, l\rangle \in \mathcal{L}$.

Then a *network representation* Δ is any triple $\langle \mathcal{N}, \mathcal{A}, \mathcal{L}\rangle$ where

- $\mathcal{N}$ is a set of names, as before; we now use $\mathbf{loc}(\mathcal{N})$ to represent the subset of $\mathcal{N}$ which are locations
- $\mathcal{A} \subseteq \mathbf{loc}(\mathcal{N})$ represents the set of dead locations, as before.
- $\mathcal{L} \subseteq \mathbf{loc}(\mathcal{N}) \times \mathbf{loc}(\mathcal{N})$ represents the set of connections between locations ■

As with DπLoc network representations, we use the notation $\Delta_{\mathcal{N}}$, $\Delta_{\mathcal{A}}$ and $\Delta_{\mathcal{L}}$ to refer to the individual components of Δ. We will also have various notation for checking properties of DπF network representations, and updating them; these will be explained informally, with the formal definitions relegated to the Appendix.

3.2 Reduction Semantics of DπF

The definition of well-formed configurations, Definition 2.2.1 generalises in a straightforward manner: we say $\Delta \triangleright M$ is a well-formed configuration if every free name occuring in M is also in $\Delta_{\mathcal{N}}$. Then the judgements of the reduction semantics take the form

$$\Delta \triangleright M \longrightarrow \Delta' \triangleright M'$$

where $\Delta \triangleright M$ and $\Delta' \triangleright M'$ are well-formed configurations. This is defined as the least relation which satisfies the rules in Table 2 and Table 4 (substituting Δ for Π) , all inherited

Table 10. *New Network Reduction Rules for DπF*

Assuming $\Delta \vdash l : \textbf{alive}$

(r-go)
$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Delta \triangleright k[\![P]\!]}\ \Delta \vdash k \leftarrow l$$

(r-ngo)
$$\frac{}{\Delta \triangleright l[\![\mathsf{go}\, k.P]\!] \longrightarrow \Delta \triangleright k[\![\mathbf{0}]\!]}\ \Delta \nvdash k \leftarrow l$$

(r-ping)
$$\frac{}{\Delta \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Delta \triangleright l[\![P]\!]}\ \Delta \vdash k \leftarrow l$$

(r-nping)
$$\frac{}{\Delta \triangleright l[\![\mathsf{ping}\, k.P\lceil Q\rceil]\!] \longrightarrow \Delta \triangleright l[\![Q]\!]}\ \Delta \nvdash k \leftarrow l$$

(r-newc)
$$\frac{}{\Delta \triangleright l[\![(\nu\, c : \mathsf{ch})\, P]\!] \longrightarrow \Delta \triangleright (\nu\, c : \mathsf{ch})\, l[\![P]\!]}$$

(r-newl)
$$\frac{}{\Delta \triangleright l[\![(\nu\, k : \mathtt{loc[S,C]})\, P]\!] \longrightarrow \Delta \triangleright (\nu\, k : \mathtt{loc[S,D]})\, l[\![P]\!]}\ \mathtt{loc[S,D]} = \mathsf{inst}(\mathtt{loc[S,C]}, l, \Delta)$$

(r-kill)
$$\frac{}{\Delta \triangleright l[\![\mathsf{kill}]\!] \longrightarrow (\Delta - l) \triangleright l[\![\mathbf{0}]\!]}$$

(r-brk)
$$\frac{}{\Delta \triangleright l[\![\mathsf{break}\, k]\!] \longrightarrow (\Delta - l \leftrightarrow k) \triangleright l[\![\mathbf{0}]\!]}\ \Delta \vdash l \leftrightarrow k$$

from the reduction rules for DπLoc, together with the new reduction rules of Table 10, which we now explain. We note that, as usual, all of these rules require that the location where the activity is to occur is alive, $\Delta \vdash l : \textbf{alive}$.

The most subtle but important changes to the network reductions rules are those concerning the constructs go and ping. Even though the general intuition remains the same to that of § 2.2, the former notion of k being accessible from l, used by rules such as (r-go) and (r-nping), and still denoted as $\Delta \vdash k \leftarrow l$, changes; for k to be *accessible* from l, two conditions must hold, namely that k is alive *and* that the link between l and k is alive as well. If any of these two conditions do not hold, then k is deemed to be *inaccessible* from the point of view of l, denoted as before as $\Delta \nvdash k \leftarrow l$. The more complex network representation has also an impact on the information that can be gathered by the construct ping; in DπLoc, if ping reduced using (r-nping), it meant that the location being tested for was dead; in DπF however, such a reduction merely means that the destination is inaccessible, which could be caused by a dead destination location, a broken link to the destination location or both.

At this point we note that, in DπF, since *not* every node is interconnected, it makes more sense to talk about *reachability* rather *accessibility* between nodes. A node k is *reachable* from l in Δ, denoted as $\Delta \vdash k \leftarrow\!\!\!\rightsquigarrow l$, if it is accessible using one or more migrations; the

Table 11. *New Structural Rules for DπF*

...				
(s-flip-1)	$(\nu\, n{:}\mathtt{T})(\nu\, m{:}\mathtt{U})N$	$\equiv$	$(\nu\, m{:}\mathtt{U})(\nu\, n{:}\mathtt{T})N$	$n \notin \mathbf{fn}(\mathtt{U})$
(s-flip-2)	$(\nu\, n{:}\mathtt{T})(\nu\, m{:}\mathtt{U})N$	$\equiv$	$(\nu\, m : \mathtt{U}{-}n)(\nu\, n : \mathtt{T}{+}m)N$	$n \in \mathbf{fn}(\mathtt{U}),\ m \notin \mathbf{fn}(\mathtt{T})$
...				

formal definition is relegated to the Appendix.

The other main change in Table 10 is the rule for creating new locations, (r-newl); here, the links to the new location k need to be calculated and the network Δ updated. This is achieved by the function $\mathsf{inst}(\mathtt{T}, l, \Delta)$, the formal definition of which is:

$$\mathsf{inst}(\mathtt{T}, l, \Delta) \stackrel{\text{def}}{=} \begin{cases} \mathtt{loc}[\{k \mid k \in \mathtt{C} \text{ and } \Delta \vdash k \leftrightsquigarrow l\} \cup \{l\}] & \text{if } \mathtt{T} = \mathtt{loc}[\mathtt{a}, \mathtt{C}] \\ \mathtt{T} & \text{otherwise} \end{cases}$$

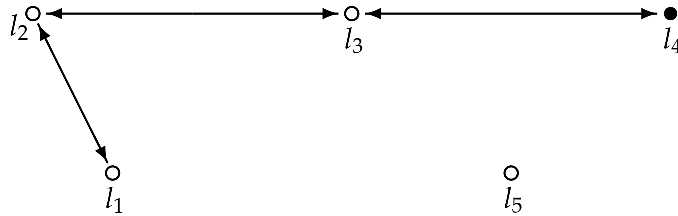
Intuitively $\mathsf{inst}(\mathtt{loc}[\mathtt{S}, \mathtt{C}], l, \Delta)$, for the case where $\mathtt{S} = \mathtt{a}$, returns the location type $\mathtt{loc}[\mathtt{S}, \mathtt{D}]$, where the set of locations $\mathtt{D}$, is the subset of locations in $\mathtt{C} \cup \{l\}$ which are *reachable* from l; this construction is further explained in Example 3.2.1 below. The final new rule in Table 10 is (r-brk), simulating the breaking of a link; the intuition behind the network updating function $\Delta - l \leftrightarrow k$ should be obvious.

To complete the reduction semantics of DπF we need to revise the rules in Table 5, defining the structural equivalence. The revision is detailed in Table 11; the rule (s-flip) is replaced by the two rules (s-flip-1) and (s-flip-2). This enables us to flip two successively scoped locations even if the first is used in the type of the second, that is there is a link between the two scoped locations.

Example 3.2.1. Consider the system:

$$\mathsf{launchNewLoc} \Leftarrow l_3[\![a!\langle l_1\rangle]\!] \mid l_3[\![a?(x).(\nu k : \mathtt{loc}[\mathtt{a}, \{x, l_2, l_4, l_5\}])P]\!]$$

running on a network Δ consisting of five locations $l_1..l_5$, all of which are alive except l_4, with l_2 connected to l_1 and l_3, and l_3 connected to l_4. Diagrammatically this is easily represented as:



where, open nodes (∘) represent live locations and closed ones (•) dead locations; we systematically omit reflexive links in these network diagrams. Formally describing Δ is more tedious:

- $\Delta_{\mathcal{N}}$ is $\{a,\ l_1,\ l_2,\ l_3,\ l_4,\ l_5\}$
- $\Delta_{\mathcal{A}}$ is $\{l_1,\ l_2,\ l_3,\ l_5\}$
- the link set $\Delta_{\mathcal{L}}$ is given by

$$\left\{\begin{array}{l}\langle l_1,l_1\rangle,\ \langle l_2,l_2\rangle,\ \langle l_3,l_3\rangle,\ \langle l_4,l_4\rangle,\ \langle l_5,l_5\rangle,\\ \langle l_1,l_2\rangle,\ \langle l_2,l_3\rangle,\ \langle l_3,l_4\rangle,\ \langle l_2,l_1\rangle,\ \langle l_3,l_2\rangle,\langle l_4,l_3\rangle\end{array}\right\}$$

Clearly there is considerable redundancy in this representation of link sets; $\Delta_{\mathcal{L}}$ can be more reasonably represented as:

$$\Delta_{\mathcal{L}}\ =\ \{l_1 \leftrightarrow l_2,\ l_2 \leftrightarrow l_3,\ l_3 \leftrightarrow l_4,\ l_5 \leftrightarrow l_5\}$$

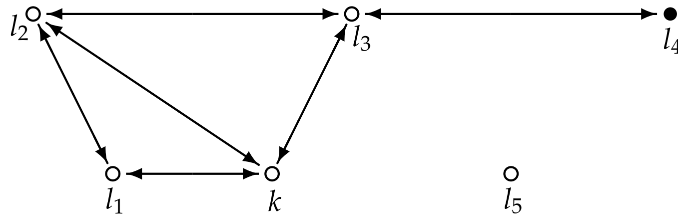
where $l \leftrightarrow k$ denotes the pair of pairs $\langle l,k\rangle$, $\langle k,l\rangle$ together with the reflexive pairs $\langle l,l\rangle$, $\langle k,k\rangle$; in such cases, a reflexive bi-direcitional link $l \leftrightarrow l$ would be used for completely disconnected nodes such as l_5. When we apply the reduction semantics to the configuration $\Delta \triangleright$ launchNewLoc, the rule (r-comm) is used first to allow the communication of the value l_1 along a, and then (r-newl) can be used to launch the declaration of k to the system level. However, the evaluation of inst(l_3, $\mathtt{loc}[\mathsf{a}, \{l_1, l_2, l_4, l_5\}], \Delta$) at launching turns out to be $\mathtt{loc}[\mathsf{a}, \{l_1, l_2, l_3\}]$ because:

- the location from where k is launched, that is l_3, is automatically connected to k.
- l_1 and l_2 are reachable from the location where the new location k is launched, that is $\Delta \vdash l_1 \leftrightsquigarrow l_3$, $l_2 \leftrightsquigarrow l_3$; l_2 is directly accessible from l_3 while l_1 is reachable indirectly through l_2
- l_4 and l_5 are not reachable from l_3; l_4 is dead and thus it is not accessible from any other node; l_5 on the other hand, is completely disconnected.

So the resulting configuration is:

$$\Delta \triangleright (\nu\, k : \mathtt{loc}[\mathsf{a}, \{l_1, l_2, l_3\}])\ \ l_3[\![P\{^{l_1}\!/\!_x\}]\!]$$

The network Δ of course does not change, but if we focus on the system $l_3[\![P\{^{l_1}\!/\!_x\}]\!]$, we see that it is running on the internal network represented by:



■

This distinction between the internal networks used by different subsystems has already occurred in the semantics of DπLoc; see the discussion of Example 2.2.2. Nevertheless, we warn to the reader that there will be more serious consequences for DπF, due to the complex nature of reachability that comes into play.

3.2.1 Reduction barbed congruence

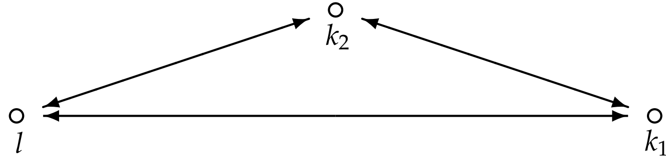
The definition of Reduction barbed congruence, Definition 2.2.6, originally developed for DπLoc configurations, can be adapted to apply also to DπF. The formal definition is delayed to Section 3.5, but let us use the the same notation,

$$\Delta \models M \cong N \tag{3.1}$$

to indicate that the systems M and N are equivalent relative to the network Δ; the following discussion relies on an intuitive understanding of this concept.

Let us now reconsider the three implementations of a client server discussed in Example 2.2.9, but this time running on a network with explicit links. For convenience, in this and later examples, we systematically omit channel names from network representations. Moreover, we abbreviate the location type `loc[a,C]` to `loc[C]` when the status of location is understood to be alive.

Example 3.2.2. Let Δ represent the following network:



Formally Δ is determined by letting $\Delta_{\mathcal{N}}$ be $\{l, k_1, k_2\}$, $\Delta_{\mathcal{A}}$ be $\emptyset$ and $\Delta_{\mathcal{L}}$ be $\{l \leftrightarrow k_1,\ l \leftrightarrow k_2,\ k_1 \leftrightarrow k_2\}$.

The distributed server implementations, srvDis and srv2Rt, presented earlier in Example 2.2.9, are no longer reduction barbed congruent relative to Δ, because in this extended setting, the behaviour of systems is also examined in the context of faulty links. It is sufficient to consider the possible barbs in the context of a client such as $l[\![req!\langle l, ret\rangle]\!]$ and a fault inducing context:

$$C_3 \ = \ [-] \,|\, l[\![\mathsf{break}\ k_1]\!]$$

which breaks the link $l \leftrightarrow k_1$. Stated otherwise, if the link $l \leftrightarrow k_1$ breaks, srv2Rt will still be able to operate normally and barb on *ret@l*; srvDis, on the other hand, may reach a state where it blocks since migrating directly back and forth from l to k_1 becomes prohibited and as a result, it would not be able to barb *ret@l*. However consider the alternative remote client srvMtr, defined as:

$$\mathsf{srvMtr} \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu sync)\left(\begin{array}{l}\mathsf{go}\ k_1.\,data!\langle x, sync\rangle \\ |\,\mathsf{monitor}\ k_1\lceil \mathsf{go}\ k_2, k_1.data!\langle x, sync\rangle\rceil \\ |\,sync?(x).y!\langle x\rangle \end{array}\right)\right]\!\!\right] \\ \\ |\,k_1\left[\!\!\left[data?(x,y).\left(\begin{array}{l}\mathsf{go}\ l.\,y!\langle f(x)\rangle \\ |\,\mathsf{monitor}\ l\lceil \mathsf{go}\ k_2, l.y!\langle f(x)\rangle\rceil\end{array}\right)\right]\!\!\right]\end{array}\right)$$

where the macro monitor $k\lceil P\rceil$, is a process that repeatedly tests the accessibility of a location k from the hosting location, and launches P when k becomes inaccessible. It is

formally defined as:

$$\text{monitor}\ k\lceil P \rceil \;\Leftarrow\; (\nu\, test\!:\!\text{ch})(\, test!\langle\rangle \mid *\, test?().\text{ping}\ k.\ test!\langle\rangle\lceil P \rceil\,)$$

It turns out that $\Delta \models \text{srv2Rt} \cong \text{srvMntr}$ but once again, it is difficult to establish because of a typical formulation of reduction barbed congruence would quantify over all possible contexts. ■

In the next example we examine the interplay between dead nodes and dead links.

Example 3.2.3. Consider the following three networks,

$$\Delta_1 \;=\; \Delta_l + k\!:\!\texttt{loc}[\texttt{d}, \{l\}] \;=$$

$$\Delta_2 \;=\; \Delta_l + k\!:\!\texttt{loc}[\texttt{d}, \emptyset] \;=$$

$$\Delta_3 \;=\; \Delta_l + k\!:\!\texttt{loc}[\texttt{a}, \emptyset] \;=$$

These are the effective networks for the system $l[\![a!\langle k\rangle]\!]$ in the three configurations $\Delta_l \triangleright N_i$, where N_i are defined by

$$N_1 \;\Leftarrow\; (\nu k : \texttt{loc}[\texttt{d}, \{l\}])\ l[\![a!\langle k\rangle]\!]$$
$$N_2 \;\Leftarrow\; (\nu k : \texttt{loc}[\texttt{d}, \emptyset])\ l[\![a!\langle k\rangle]\!]$$
$$N_3 \;\Leftarrow\; (\nu k : \texttt{loc}[\texttt{a}, \emptyset])\ l[\![a!\langle k\rangle]\!]$$

and Δ_l is the simple network with one live location l:

$$\Delta_l \;=\; \langle \{l, a\}, \emptyset, \{l \leftrightarrow l\}\rangle$$

Intuitively, no observer can distinguish between these three configurations; even though some observer might obtain the scoped name k via the channel a at l, it cannot determine the difference in the state of the network. From rule (l-nmove) we conclude that any attempt to move from l, where the observer would be located, to k will fail. However, such a failure does not yield the observer enough information to determine the exact nature of the fault causing the failure: the observer holding k does not know whether the inaccessibility failure to k was caused by a node fault at k, a link fault between l and k or both. As we shall see later, we will be able to demonstrate $\Delta_l \models N_1 \cong N_2 \cong N_3$. ■

3.3 A labelled transition system for DπF

It would be tempting to mimic the development of Section 2.3 and define a bisimulation equivalence based on actions of the form

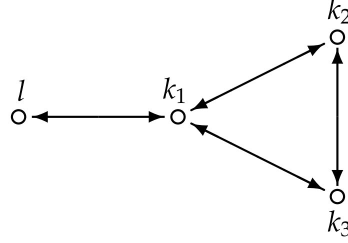
$$\Delta \triangleright M \xrightarrow{\mu} \Delta' \triangleright M'$$

Here we argue that this would not be adequate, at least if the target is to characterise reduction barbed congruence.

Example 3.3.1. Let Δ_l be the network in which there is only one node l which is alive, defined earlier in Example 3.2.3, and consider the system:

$$M_1 \quad \Leftarrow \quad (\nu\, k_1 : \{l\})\,(\nu\, k_2 : \{k_1\})\,(\nu\, k_3 : \{k_1, k_2\})\, l[\![a!\langle k_2, k_3\rangle.P]\!]$$

Note that when M_1 is running on Δ_l, due to the new locations declared, the code $l[\![a!\langle k_2, k_3\rangle.P]\!]$ is effectively running on the following *internal* network (part of which is scoped):



(3.2)

Let us now see to what knowledge of this internal network can be gained by an observer O at site l, such as $l[\![a?(x,y).O(x,y)]\!]$. Note, that prior to any interaction, O is running on the network Δ_l, and thus, is only aware of the unique location l. By inputting along a, it can gain knowledge of the two names k_2 and k_3, thereby evolving to $l[\![O(k_2, k_3)]\!]$. Yet, even though it is in possession of these two names, it cannot move there and interact with code at k_2 and k_3. It can niether discover the link between k_2 and k_3, due to the fact that it is not aware of the (scoped) name k_1. The scoping of k_1 prohibits the observer from discovering the full extent of the internal network (3.2) above since, with its current knowledge of location names, it cannot construct a path to reach either k_2 or k_3.

This means that, there is now a difference between the actual network being used by the system, (3.2), and the observer's *view* of that network. Even worse, the current formalism does not allow us to represent this (*external*) observer view of the network. ■

An lts semantics will have to record the differences between the network and the observers view of networks. This requires extra information being recorded in network representations.

Definition 3.3.2 (Effective network representations). An *effective network representation* Σ is a triple $\langle \mathcal{N}, \mathcal{O}, \mathcal{H}\rangle$, where:

- $\mathcal{N}$ is a set of names, as before, divided into $\mathbf{loc}(\mathcal{N})$ and $\mathbf{chan}(\mathcal{N})$,
- $\mathcal{O}$ is a *linkset*, denoting the live locations and links that are *observable* by the context.
- $\mathcal{H}$ is another *linkset*, denoting the live locations and links that are *hidden* (or unreachable) to the context.

The only consistency requirements are that:

1. $\mathbf{dom}(\mathcal{O}) \subseteq \mathbf{loc}(\mathcal{N})$ (the observable live state concerns locations in $\mathcal{N}$)
2. $\mathbf{dom}(\mathcal{H}) \subseteq \mathbf{loc}(\mathcal{N})$ (the hidden live state concerns locations in $\mathcal{N}$)

3. $\mathbf{dom}(O) \cap \mathbf{dom}(\mathcal{H}) = \emptyset$ (live state cannot be both observable and hidden) ■

The intuition is that an observer running on a network representation Σ, knows about all the names in $\Sigma_{\mathcal{N}}$ and has access to all the locations in $\mathbf{dom}(O)$; as a result, it knows the state of every location in $\mathbf{dom}(O)$ and the live links between these locations. The observer, however, does not have access to the live locations in $\mathbf{dom}(\mathcal{H})$; as a result, it cannot determine the live links between them nor can it distinguish them from dead nodes. Dead nodes are encoded in Σ as $\mathbf{loc}(\mathcal{N})/\mathbf{dom}(O \cup \mathcal{H})$, that is, all the location names in $\mathcal{N}$ that are not mentioned in either O or $\mathcal{H}$; these are conveniently denoted as the deadset $\Sigma_{\mathcal{D}}$. We also note that the effective network representation Σ does not represent live links where either end point is a dead node, since these can never be used nor observed. Summarising, Σ hold all the necessary information from the observer's point of view, that is, the names known, $\mathcal{N}$, the state known, O, and the state that can potentially become known in future, as a result of scope extrusion, $\mathcal{H}$.

As usual we use notation such as $\Sigma_{\mathcal{N}}$, Σ_O and $\Sigma_{\mathcal{H}}$ to access the fields of Σ and note that any network representation Δ can be translated into an extended network representation $\Sigma(\Delta)$ in the obvious manner:

- the set of names remains unchanged, $\Sigma(\Delta)_{\mathcal{N}} = \Delta_{\mathcal{N}}$
- the accessible state and connections, $\Sigma(\Delta)_O$, is simply $\Delta_{\mathcal{L}}$ less the dead nodes and links to these dead nodes, thus denotes as $\Delta_{\mathcal{L}}/\Delta_{\mathcal{D}}$.
- the hidden state, $\Sigma(\Delta)_{\mathcal{H}}$, is simply the empty set, since Δ does not encode any inaccessible live locations to the observer.

There is also an obvious operation for reducing an extended network representation Σ into a standard one, yielding: $\Delta(\Sigma)$:

- $\Delta(\Sigma)_{\mathcal{N}}$ is inherited directly from Σ.
- $\Delta(\Sigma)_{\mathcal{A}}$ is $\mathbf{dom}(\Sigma_O \cup \Sigma_{\mathcal{H}})$ as stated earlier.
- $\Delta(\Sigma)_{\mathcal{L}}$ is simply $\Sigma_O \cup \Sigma_{\mathcal{H}}$

We note two properties about the operation $\Delta(\Sigma)$; firstly, it does not represent any links to and between dead nodes in $\Delta(\Sigma)_{\mathcal{L}}$; secondly, it merges the accessible and inaccessible state into one single accessible state. Whenever we wish to forget about the distinction between the live accessible nodes and links in Σ and those unknown to the observer, we can transform Σ into the $\Sigma(\Delta(\Sigma))$; this we denote by $\uparrow(\Sigma)$.

For a discussion on how extended network representations allow us to accommodate the observers view in the example just discussed in Example 3.3.1, see Example 3.3.7 below.

Our lts for DπF will be defined in terms of judgements which take the form

$$\Sigma \triangleright M \xrightarrow{\mu} \Sigma' \triangleright M' \tag{3.3}$$

where the actions μ are the same as those used in the previous section, and both $\Sigma \triangleright M$ (and $\Sigma' \triangleright M'$) is an *effective* configuration, that is all the free names in M occur in $\Sigma_{\mathcal{N}}$. As stated earlier, in the configuration $\Sigma \triangleright M$, where Σ is the effective network $\langle \mathcal{N}, \mathcal{O}, \mathcal{H} \rangle$, only the information in $\mathcal{N}$ and $\mathcal{O}$ is available to an external observer, while the extra liveness information in $\mathcal{H}$ is only available internally to the system M. This division makes more complicated the various operations for extracting information from, and extending networks. As usual, all the formal definitions are relegated to the Appendix, but it is necessary to go into some detail as to how effective networks are augmented with a new location. This will have to take into account the type of the new location, and in particular the existing locations to which it will be linked. For instance, the declaration of the new location $k : \mathtt{loc}[\mathtt{a}, \mathtt{C}]$, requires adding to the network a new live location k, linked to every live location in $\mathtt{C}$.

To simplify the task of defining effective network augmentations, we first define a special form of linkset called *components* together with its related notation, and then express network augmentations in terms of this definition.

Definition 3.3.3 (Component Linksets). We start by adapting the notion of reachability in a network, $\Delta \vdash k \leftrightsquigarrow l$, to linksets, now denoted as $\mathcal{L} \vdash k \leftrightsquigarrow l$. Thus for any linkset $\mathcal{L}$:

- $\mathcal{L} \vdash k \leftarrow l \stackrel{\text{def}}{=} \langle l, k \rangle \in \mathcal{L}$
- $\mathcal{L} \vdash k \leftrightsquigarrow l \stackrel{\text{def}}{=} \mathcal{L} \vdash k \leftarrow l$ or $\exists k'$ such that $\mathcal{L} \vdash k' \leftarrow l$ and $\mathcal{L} \vdash k \leftrightsquigarrow k'$

Based on this intuition, a *component linkset* (or component), denoted by $\mathcal{K}$, is a linkset that is *completely connected*, that is:

$$\forall\, l, k \in \mathbf{dom}(\mathcal{L}) \text{ we have } \mathcal{L} \vdash k \leftrightsquigarrow l$$

We note that any linkset $\mathcal{L}$ can be treated as the union of one or more components, that is:

$$\mathcal{L} = \bigcup_{i=1}^{n} \mathcal{K}_i$$

According to such treatment, a location $l \in \mathbf{dom}(\mathcal{L})$ can be used to identify a particular component $\mathcal{K}$ in the linkset $\mathcal{L}$. We use $\mathcal{L} \leftrightsquigarrow l$ to denote the component in $\mathcal{L}$ identified by a location l and this formally is defined as:

$$\mathcal{L} \leftrightsquigarrow l \stackrel{\text{def}}{=} \{\langle k, k' \rangle \mid \langle k, k' \rangle \in \mathcal{L} \text{ and } \mathcal{L} \vdash k \leftrightsquigarrow l\}$$

Similarly, a set of locations $\{l_1, \ldots, l_n\}$ can identify a number of components in $\mathcal{L}$, denoted and defined as:

$$\mathcal{L} \leftrightsquigarrow \{l_1, \ldots, l_n\} \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \mathcal{L} \leftrightsquigarrow l_i$$

Finally, if $C = \{k_1, \ldots, k_n\}$ is a set of locations representing connections, and l is a location such that $l \notin C$, then $l \leftrightarrow C$ denotes the component defined as:

$$l \leftrightarrow C \stackrel{\text{def}}{=} \{\langle k, l \rangle, \langle l, k \rangle, \langle k, k \rangle \mid k \in C\} \cup \{\langle l, l \rangle\}$$

where locations in C are symmetrically related to l, while l is also related to itself. In the resultant component $l \leftrightarrow C$, all the locations in C in the component $l \leftrightarrow C$ are connected as a star formation to l and as a result, all locations are reachable from one another in *at most* two accesses by going through the central node l. Using previous shorthand notation, we could have alternatively defined $l \leftrightarrow C$ as:

$$l \leftrightarrow C \stackrel{\text{def}}{=} \{l \leftrightarrow k, \mid k \in C\}$$

■

Lemma 3.3.4 (Subtracting a Component from a Linkset). For any linkset $\mathcal{L}$ and component $\mathcal{K}$ such that $\mathcal{K} \subseteq \mathcal{L}$, the set $\mathcal{L}/\mathcal{K}$ is also a linkset.

Proof. Immediate from the fact that $\mathcal{L}$ can be expressed as $\bigcup_{i=1}^{n} \mathcal{K}_i$ where $\mathcal{K}$ must be equal to one $\mathcal{K}_i$. Thus, if $\mathcal{K} = \mathcal{K}_j$, the set $\bigcup_{j \neq i=1}^{n} \mathcal{K}_i$, which translates to $\mathcal{L}/\mathcal{K}$, would still be a linkset. □

We now revert our discussion back to effective network representation, and show how the definition of components facilitates the procedure for extending networks.

Definition 3.3.5 (Augmenting effective networks). Let n be fresh to the network Σ and $\mathtt{C}$ be a set of locations such that $\mathtt{C} \subseteq \mathbf{dom}(\Sigma_{O})$. Then we define the operation $\Sigma + n : \mathtt{T}$ as:

- $\Sigma + n : \mathtt{ch} \stackrel{\text{def}}{=} \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{O},\ \Sigma_{\mathcal{H}} \rangle$
- $\Sigma + n : \mathtt{loc[d, C]} \stackrel{\text{def}}{=} \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{O},\ \Sigma_{\mathcal{H}} \rangle$
- $\Sigma + n : \mathtt{loc[a, C]} \stackrel{\text{def}}{=}$

$$\begin{array}{llll} \text{Case} & \mathtt{C} \cap \mathbf{dom}(\Sigma_{O}) = \emptyset & \text{then} & \langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{O},\ \mathcal{H}' \rangle \\ & & & \text{where: } \mathcal{H}' = \Sigma_{\mathcal{H}} \cup (l \leftrightarrow \mathtt{C}) \\ & \mathtt{C} \cap \mathbf{dom}(\Sigma_{O}) \neq \emptyset & \text{then} & \langle \Sigma_{\mathcal{N}} \cup \{n\},\ O',\ \mathcal{H}' \rangle \\ & & & \text{where: } O' = \Sigma_{O} \cup (l \leftrightarrow \mathtt{C}) \cup (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}) \\ & & & \quad \text{and } \mathcal{H}' = \Sigma_{\mathcal{H}}/(\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}) \end{array}$$ ■

In the above definition, extending a network with a fresh channel is trivial; adding a fresh dead node is similarly simple, due to the fact that Σ does not represent dead nodes or links to dead nodes explicitly. The only subcase that deserves some explanation is that of adding fresh *live* nodes. A fresh live location is added to either Σ_{O} or $\Sigma_{\mathcal{H}}$ depending on its links. If it is not linked to any observable location, $\mathtt{C} \cap \mathbf{dom}(\Sigma_{O}) = \emptyset$, then the new fresh location is not reachable from the context and is therefore added to $\Sigma_{\mathcal{H}}$. If, on the other hand, it is linked to an observable location, $\mathtt{C} \cap \mathbf{dom}(\Sigma_{O}) \neq \emptyset$, then it becomes observable as well. There is also the case where the fresh location is linked to both observable and hidden locations, still represented above by the case where $\mathtt{C} \cap \mathbf{dom}(\Sigma_{O}) \neq \emptyset$; in such a case, the fresh location, together with any components in the hidden state linked to it, that is $\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}$, become observable and thus transfered from $\Sigma_{\mathcal{H}}$ to Σ_{O}. The following example elucidates this operation for extending effective networks.

Example 3.3.6. Consider the effective network Σ, representing six locations $l, k_1, \ldots, k_5$:

$$\Sigma = \langle \{l, k_1, k_2, k_3, k_4, k_5\},\ \{l \leftrightarrow l\},\ \{k_1 \leftrightarrow k_2, k_2 \leftrightarrow k_3, k_4 \leftrightarrow k_4\} \rangle$$

According to Definition 3.3.2, l is the only observable location by the context; locations $k_{1..4}$ are alive but not reachable from any observable location while the remaining location, k_5, is dead since it is not in $\mathbf{dom}(\Sigma_{\mathcal{O}} \cup \mathcal{H})$. Moreover, the linkset representing the hidden state, $\Sigma_{\mathcal{H}}$, can be partitioned into two components, $\mathcal{K}_1 = \{k_1 \leftrightarrow k_2, k_2 \leftrightarrow k_3\}$ and $\mathcal{K}_2 = \{k_4 \leftrightarrow k_4\}$ whereas the linkset representing the observable state, $\Sigma_{\mathcal{O}}$, can only be partitioned into one component, itself.

The operation $\Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \{l\}]$ would make the fresh location, k_0, observable in the resultant effective network since it is linked to, thus reachable from, the observable location l. On the other hand, the operation $\Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \emptyset]$ would make k_0 hidden since it is a completely disconnected node, just like k_4. The operation $\Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \{k_1\}]$ would still make k_0 hidden in the resultant effective network, since it is only link to the hidden node k_1. Finally, the operation $\Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \{l, k_1\}]$ intersects with both $\Sigma_{\mathcal{O}}$ and $\Sigma_{\mathcal{H}}$. This means that k_0 itself becomes observable, but as a side effect, the components reachable through it, that is $\Sigma_{\mathcal{H}} \leftsquigarrow\!\!\rightsquigarrow \{l, k_1\} = \mathcal{K}_1$, becomes observable as well. Thus, according to Definition 3.3.5, the updated network translates to:

$$\begin{array}{llll}
\Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \{l, k_1\}] = & & & \\
\langle\ \Sigma_{\mathcal{N}} \cup \{k_0\}, & \Sigma_{\mathcal{O}} \cup (k_0 \leftrightarrow \{l, k_1\}) \cup (\Sigma_{\mathcal{H}} \leftsquigarrow\!\!\rightsquigarrow \{l, k_1\}), & \Sigma_{\mathcal{H}} / (\Sigma_{\mathcal{H}} \leftsquigarrow\!\!\rightsquigarrow \{l, k_1\}) & \rangle \\
\langle\ \Sigma_{\mathcal{N}} \cup \{k_0\}, & \Sigma_{\mathcal{O}} \cup \{k_0 \leftrightarrow l, k_0 \leftrightarrow k_1\} \cup \mathcal{K}_1, & \Sigma_{\mathcal{H}} / \mathcal{K}_1 & \rangle \\
\langle\ \Sigma_{\mathcal{N}} \cup \{k_0\}, & \{l \leftrightarrow k_0, k_0 \leftrightarrow k_1, k_1 \leftrightarrow k_2, k_2 \leftrightarrow k_3\}, & \{k_4 \leftrightarrow k_4\} & \rangle
\end{array}$$

■

Let us now return to the defintion of our lts for DπF. The transitions between effective configurations (3.3) are determined by the rules and axioms already given in Table 6 from Section 2.3, together with the new rules in Table 12 and Table 13. Most of the rules in Table 12 are inherited directly from their counterpart reduction rules in Table 10. The new rule is (l-disc) which introduces the new label $l \nleftrightarrow k$ and models the breaking of a link from the observing context, in the same fashion as (l-fail) in Table 8 models external location killing. Both of these rules are subject to the condition that the location or link where the fault is injected is observable by the context, that is $\Sigma \vdash_{\mathsf{obs}} l : \mathbf{alive}$ and $\Sigma \vdash_{\mathsf{obs}} l \leftrightarrow k$ respectively. We also note that rule (l-newl) is similar to its corresponding reduction rule (r-newl), but now we overload the function $\mathsf{inst}(\mathtt{loc}[\mathtt{S}, \mathtt{C}], l, \Sigma)$, previously defined over Δ, to the new network representation Σ.

The more challenging rules are found in Table 13. Most of these are slightly more subtle versions of the corresponding rules for DπLoc in Table 8; the subtleties are required to deal with the interaction between scoped location names and their occurrence in location types. For instance, the rule (l-open) filters the type of scope extruded locations by removing links to locations that are already dead and that will not affect the effective network Σ; this is done through the operation $\mathtt{T}/\Sigma_{\mathcal{D}}$ defined in the Appendix. A side condition is added to

Table 12. *Network Operational Rules(2) for DπF*

Assuming $\Sigma \vdash l : \textbf{alive}$

(l-kill)
$$\frac{}{\Sigma \triangleright l[\![\text{kill}]\!] \xrightarrow{\tau} (\Sigma - l) \triangleright l[\![\mathbf{0}]\!]}$$

(l-brk)
$$\frac{}{\Sigma \triangleright l[\![\text{break } k]\!] \xrightarrow{\tau} \Sigma - (l \leftrightarrow k) \triangleright l[\![\mathbf{0}]\!]} \; \Sigma \vdash l \leftrightarrow k$$

(l-halt)
$$\frac{}{\Sigma \triangleright N \xrightarrow{\text{kill}:l} (\Sigma - l) \triangleright N} \; \Sigma \vdash_{\text{obs}} l : \textbf{alive}$$

(l-disc)
$$\frac{}{\Sigma \triangleright N \xrightarrow{l \not\leftrightarrow k} \Sigma - (l \leftrightarrow k) \triangleright N} \; \Sigma \vdash_{\text{obs}} l \leftrightarrow k$$

(l-go)
$$\frac{}{\Sigma \triangleright l[\![\text{go } k.P]\!] \xrightarrow{\tau} \Sigma \triangleright k[\![P]\!]} \; \Sigma \vdash k \leftarrow l$$

(l-ngo)
$$\frac{}{\Sigma \triangleright l[\![\text{go } k.P]\!] \xrightarrow{\tau} \Sigma \triangleright k[\![\mathbf{0}]\!]} \; \Sigma \nvdash k \leftarrow l$$

(l-ping)
$$\frac{}{\Sigma \triangleright l[\![\text{ping } k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![P]\!]} \; \Sigma \vdash k \leftarrow l$$

(l-nping)
$$\frac{}{\Sigma \triangleright l[\![\text{ping } k.P\lceil Q \rceil]\!] \xrightarrow{\tau} \Sigma \triangleright l[\![Q]\!]} \; \Sigma \nvdash k \leftarrow l$$

(l-newc)
$$\frac{}{\Sigma \triangleright l[\![(\nu\, c:\text{ch})\, P]\!] \xrightarrow{\tau} \Sigma \triangleright (\nu\, c:\text{ch})\, l[\![P]\!]}$$

(l-newl)
$$\frac{}{\Sigma \triangleright l[\![(\nu\, k:\texttt{loc[S,C]})\, P]\!] \xrightarrow{\tau} \Sigma \triangleright (\nu\, k:\texttt{loc[S,D]})\, l[\![P]\!]} \; \texttt{loc[S,D]} = \text{inst}(\texttt{loc[S,C]}, l, \Sigma)$$

(l-weak), $(\Sigma + \tilde{n} : \tilde{\texttt{T}}) \vdash_{\text{obs}} \texttt{T}$, limiting the types of imported fresh locations to only contain locations which are externally accessible, since intuitively, the context can only introduce fresh locations linked to locations it can access. The internal communication rule (l-par-comm) also changes slightly from the one given earlier for DπLoc; communication is defined in terms of the system view ($\uparrow(\Sigma)$) rather than the observer view dictated by Σ. The intuition for this alteration is that internal communication can still occur, even at locations that the observer cannot access, thus we denote the ability to output and input of systems with respect to the maximal observer view $\uparrow(\Sigma)$. Finally, a completely new rule is (l-rest-typ), which restricts the links exported in location types if one endpoint of the link is still scoped. The utility of this rule is illustrated further in the following example. The rules (l-rest) and (l-par-ctxt) remain unchanged for DπLoc.

Example 3.3.7. Let us revisit Example 3.3.1 to see how the effect of the observer O on M_1, running on the effective network Σ_l having only one location l which is alive, that is $\Sigma(\Delta_l)$. This effectively means calculating the result of M_1 performing an output on a at l.

It is easy to see that an application of (l-out), followed by two applications of (l-open) gives

$$\Sigma_l + k_1 : \{l\} \triangleright M_1' \quad \xrightarrow{\alpha} \quad \Sigma_l + k_1 : \{l\} + k_2 : \{k_1\} + k_3 : \{k_1, k_2\} \triangleright l[\![P]\!] \tag{3.4}$$

Table 13. *Contextual Operational Rules(3) for DπF*

(l-open)

$$\frac{\Sigma + n:\mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'}{\Sigma \triangleright (\nu\, n:\mathtt{T})N \xrightarrow{(n:\mathtt{U},\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'}\; l,a \neq n \in V,\ \mathtt{U} = \mathtt{T}/\Sigma_{\mathcal{A}}$$

(l-weak)

$$\frac{\Sigma + n:\mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'}{\Sigma \triangleright N \xrightarrow{(n:\mathtt{T},\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'}\; l,a \neq n \in V,\ (\Sigma + \tilde{n}:\tilde{\mathtt{T}}) \vdash_{\mathsf{obs}} \mathtt{T}$$

(l-rest-typ)

$$\frac{\Sigma + k:\mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} (\Sigma + \tilde{n}:\tilde{\mathtt{U}}) + k:\mathtt{U} \triangleright N'}{\Sigma \triangleright (\nu\, k:\mathtt{T})N \xrightarrow{(\tilde{n}:\tilde{\mathtt{U}})l:a!\langle V\rangle} \Sigma + \tilde{n}:\tilde{\mathtt{U}} \triangleright (\nu\, k:\mathtt{U})N'}\; l,a \neq k \in \mathbf{fn}(\tilde{\mathtt{T}})$$

(l-rest)

$$\frac{\Sigma + n:\mathtt{T} \triangleright N \xrightarrow{\mu} \Sigma' + n:\mathtt{U} \triangleright N'}{\Sigma \triangleright (\nu\, n:\mathtt{T})N \xrightarrow{\mu} \Sigma' \triangleright (\nu\, n:\mathtt{U})N'}\; n \notin \mathbf{fn}(\mu)$$

(l-par-ctxt)

$$\frac{\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'}{\begin{array}{c}\Sigma \triangleright N|M \xrightarrow{\mu} \Sigma' \triangleright N'|M \\ \Sigma \triangleright M|N \xrightarrow{\mu} \Sigma' \triangleright M|N'\end{array}}\; \Sigma \vdash M$$

(l-par-comm)

$$\frac{\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N' \qquad \uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Sigma'' \triangleright M'}{\begin{array}{c}\Sigma \triangleright N|M \xrightarrow{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})(N'|M') \\ \Sigma \triangleright M|N \xrightarrow{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathtt{T}})(M'|N')\end{array}}$$

where M_1' is $(\nu\, k_2:\{k_1\})(\nu\, k_3:\{k_1,k_2\})l[\![a!\langle k_2,k_3\rangle.P]\!]$ and α is the action $(k_2:\{k_1\},k_3:\{k_1,k_2\})l : a!\langle k_2,k_3\rangle$. Note that (l-rest) can not be applied to this judgement, since k_1 occurs free in the action α. However (3.4) can be re-arranged to read

$$\Sigma_l + k_1:\{l\} \triangleright M_1' \quad \xrightarrow{\alpha} \quad \Sigma_l + k_2:\emptyset + k_3:\{k_2\} + k_1:\{l,k_1,k_2\} \triangleright l[\![P]\!]$$

moving the addition of location k_1 in the reduct to the outmost position. At this point, (l-rest-typ) can be applied, to give

$$\Sigma_l \triangleright M_1 \quad \xrightarrow{\beta} \quad \Sigma_l + k_2:\emptyset + k_3:\{k_2\} \triangleright (\nu\, k_1 : \{l,k_1,k_2\})l[\![P]\!]$$

where β is the action $(k_2:\emptyset,\ k_3:\{k_2\})l : a!\langle k_2,k_3\rangle$, that is α filtered from any occurrence of k_1 in its bounded types. Note that the residual network representation, $\Sigma_l + k_2 : \emptyset + k_3 : \{k_2\}$ has a non-trivial internal network, not available to the observer. It evaluates to

$$\langle\{l,k_2,k_3\},\ \{l \leftrightarrow l\},\ \{k_2 \leftrightarrow k_3\}\rangle$$

and may be represented diagramatically by:

l $\circ$ $k_2 \circ$ ↕ $\circ k_3$

where the links of hidden components are denoted with dashed lines. ■

With these actions we can now define in the standard manner a bisimulation equivalence between configurations, which can be used as the basis for contextual reasoning. Let us write

$$\Sigma \models M \approx_{wrong} N$$

to mean that there is a (weak) bisimulation between the configurations $\Sigma \triangleright M$ and $\Sigma \triangleright N$ using the current actions. This new framework can be used to establish positive results. For example, for $\Sigma_{l,k} = \langle \{a, l, k\}, \{l \leftrightarrow k\}, \emptyset \rangle$, one can prove

$$\Sigma_{l,k} \models l[\![\mathsf{ping}\, k.\, a!\langle\rangle \lceil \mathbf{0} \rceil]\!] \approx_{wrong} k[\![\mathsf{go}\, l.a!\langle\rangle]\!]$$

by giving the relation $\mathcal{R}$ defined as:

$$\mathcal{R} = \left\{ \begin{array}{lcl} \langle \Sigma_{l,k} \triangleright M & , \ \Sigma_{l,k} \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l \triangleright M & , \ \Sigma_{l,k} - l \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - k \triangleright M & , \ \Sigma_{l,k} - k \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l \leftrightarrow k \triangleright M & , \ \Sigma_{l,k} - l \leftrightarrow k \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l, l \leftrightarrow k \triangleright M & , \ \Sigma_{l,k} - l, l \leftrightarrow k \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - k, l \leftrightarrow k \triangleright M & , \ \Sigma_{l,k} - k, l \leftrightarrow k \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l, k \triangleright M & , \ \Sigma_{l,k} - l, k \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \\ \langle \Sigma_{l,k} - l, k, l \leftrightarrow k \triangleright M & , \ \Sigma_{l,k} - l, k, l \leftrightarrow k \triangleright N \rangle & | \ \langle M, N \rangle \in \mathcal{R}_{sys} \end{array} \right\}$$

where

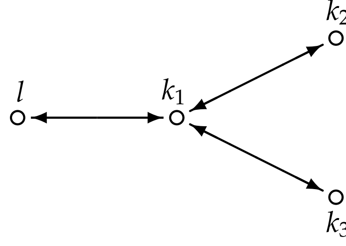
$$\mathcal{R}_{sys} = \left\{ \begin{array}{ll} \langle l[\![\mathsf{ping}\, k.\, a!\langle\rangle \lceil \mathbf{0} \rceil]\!] & , \ k[\![\mathsf{go}\, l.a!\langle\rangle]\!] \rangle \\ \langle l[\![a!\langle\rangle]\!] & , \ l[\![a!\langle\rangle]\!] \rangle \\ \langle l[\![\mathbf{0}]\!] & , \ l[\![\mathbf{0}]\!] \rangle \end{array} \right\}$$

However we can argue, at least informally, that this notion of equivalence is too discriminating and the labels too intentional, because it distinguishes between systems running on a network, where the differences in behaviour are difficult to observe. Problems arise when there is an interplay between *hidden* nodes, links and dead nodes.

Example 3.3.8. Let us consider a slight variation on the system M_1 used in Example 3.3.1 and Example 3.3.7:

$$M_2 \ \Leftarrow \ (\nu\, k_1 : \{l\})(\nu\, k_2 : \{k_1\})(\nu\, k_3 : \{k_1\}) l[\![a!\langle k_2, k_3 \rangle.P]\!]$$

again running on the simple (extended) network Σ_l. Note that here the code $l[\![a!\langle k_2, k_3 \rangle.P]\!]$ is effectively running on the following internal network,



a slight variation on that for M_1. It turns out that

$$\Sigma_l \models M_1 \not\approx_{wrong} M_2$$

because the configurations give rise to *different* output actions, on a at l. The difference lies in the types at which the locations k_2 and k_3 are exported; in $\Sigma_l \triangleright M_1$ the output label is $\mu_1 = (k_2 : \emptyset,\ k_3 : \{k_2\})\, l : a!\langle k_2, k_3\rangle$ while with $\Sigma \triangleright M_2$ it is $\mu_2 = (k_2 : \emptyset,\ k_3 : \emptyset) l : a!\langle k_2, k_3\rangle$ - there is a difference in the type associated to the scope extruded location k_3.

However if k_1 does not occur in P, (for example if P is the trivial process $\mathbf{0}$) then k_1 can never be scope extruded to the observer and thus k_2 and k_3 will remain inaccessible in both systems. This means that the presence (or absence) of the link $k_2 \leftrightarrow k_3$ can never be checked and thus there should be no observable difference between M_1 and M_2 running on Σ. ■

Problems also arise when dealing with the presence of dead nodes.

Example 3.3.9. Let us reconsider the three configurations $\Sigma_l \triangleright N_i$ for $i = 1..3$ from Example 3.2.3. We have already argued that these three configurations should not be distinguished. However, our lts specifies that all three configurations perform the output with different scope extrusion labels, namely:

$$\begin{aligned}
\Sigma_l \triangleright N_1 &\xrightarrow{(k:\mathtt{loc}[\mathtt{d},\{l\}])l:a!\langle k\rangle} \langle\{l,k\},\{l\leftrightarrow l\},\emptyset\rangle \triangleright l[\![\mathbf{0}]\!]\\
\Sigma_l \triangleright N_2 &\xrightarrow{(k:\mathtt{loc}[\mathtt{d},\emptyset])l:a!\langle k\rangle} \langle\{l,k\},\{l\leftrightarrow l\},\emptyset\rangle \triangleright l[\![\mathbf{0}]\!]\\
\Sigma_l \triangleright N_3 &\xrightarrow{(k:\mathtt{loc}[\mathtt{a},\emptyset])l:a!\langle k\rangle} \langle\{l,k\},\{l\leftrightarrow l\},\{k\leftrightarrow k\}\rangle \triangleright l[\![\mathbf{0}]\!]
\end{aligned}$$

Therefore they will be distinguished by the bisimulation equivalence which uses these actions. ■

In order to obtain a bisimulation equivalence which coincides with a typical reduction barbed congruence which qunatifies over any observer, it is necessary to abstract away from some of the information contained in the types of newly exported location names.

3.4 A bisimulation equivalence for DπF

We first outline the revision to our labelled actions. Currently these are of the form $\mathtt{T} = \mathtt{ch}$ or $\mathtt{loc}[\mathtt{A}, \{k_1, \ldots k_n\}]$, where the latter indicates the liveness of a location and the nodes k_i to which it is linked. We change these to new types of the form $\mathtt{L}, \mathtt{K} = \{l_1 \leftrightarrow k_1, \ldots, l_i \leftrightarrow k_i\}$ where $\mathtt{L}, \mathtt{K}$ are components. Intuitively, these represent the new live nodes and links, which

Table 14. *The derived lts for DπF*

(l-deriv-1)

$$\frac{\Sigma \triangleright N \xrightarrow{\mu} \Sigma' \triangleright N'}{\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'} \ \mu \in \{\tau, \mathsf{kill} : l, l \nleftrightarrow k\}$$

(l-deriv-2)

$$\frac{\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Sigma' \triangleright N'}{\Sigma \triangleright N \overset{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle}{\longmapsto} \Sigma' \triangleright N'} \ \tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$$

(l-deriv-3)

$$\frac{\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Sigma' \triangleright N'}{\Sigma \triangleright N \overset{(\tilde{n}:\tilde{\mathtt{L}})l:a?(V)}{\longmapsto} \Sigma' \triangleright N'} \ \tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$$

are made accessible to observers by the extrusion of the new location. Alternatively, this is the information which is added to the observable part of the network representation, Σ_O, as a result of the action. We have already developed the necessary technology to define these new types, in Definition 3.3.5.

Definition 3.4.1 (A (derived) labelled transition system for DπF). This consists of a collection of actions $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$, where μ takes one of the forms:

- (internal action) - τ
- (bounded input) - $(\tilde{n}:\tilde{\mathtt{L}})l : a?(V)$
- (bounded output) - $(\tilde{n}:\tilde{\mathtt{L}})l : a!\langle V\rangle$
- (external location kill) - $\mathsf{kill} : l$
- (external link break) - $l \nleftrightarrow k$ ■

The transitions in the derived lts for DπF are defined as the least relations satisfying the axioms and rules in Table 6 of Section 2.3, Tables 12 and 13 given earlier in this section and Table 14. The rules (l-deriv-2) and (l-deriv-3) transform the types of bound names using the function $\mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}}, \Sigma)$ defined below in Definition 3.4.2.

Definition 3.4.2 (Link Types). Let us first define the function $\mathsf{lnk}()$ for single typed names $n:\mathtt{T}$ and then extend it to sequences of typed names $\tilde{n}:\tilde{\mathtt{T}}$. Recalling Definition 3.3.5 for augmenting networks, the only case where $\Sigma + n:\mathtt{T}$ adds anything to the observable state of the effective network, Σ_O, is when $\mathtt{T} = \mathtt{loc}[\mathtt{a}, \mathtt{C}]$ where $\mathtt{C} \cap \mathbf{dom}(\Sigma_O) \neq \emptyset$. In such a case, we add the new location n with all its live connections denoted as $l \leftrightarrow \mathtt{C}$, and any components that were previously unreachable but have become reachable from Σ_O as a result of n, denoted as $\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}$.

When $\mathtt{C} \cap \mathbf{dom}(\Sigma_O) = \emptyset$, then the node added is unreachable for the observing contexts and we add n and its live links to $\Sigma_{\mathcal{H}}$ but nothing to Σ_O; if $\mathtt{T} = \mathtt{loc}[\mathtt{d}, \mathtt{C}]$ then we do not add anything to either Σ_O or $\Sigma_{\mathcal{H}}$ as is the case for $\mathtt{T} = \mathtt{ch}$. Based on this definition of $\Sigma + n:\mathtt{T}$, we give the following definition for $\mathsf{lnk}(n:\mathtt{T}, \Sigma)$:

$$\mathsf{lnk}(n:\mathtt{T},\Sigma) \stackrel{\text{def}}{=} \begin{cases} (n \leftrightarrow \mathsf{C}) \cup (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathsf{C}) & \text{if } \mathtt{T} = \mathtt{loc}[\mathsf{a},\mathsf{C}] \text{ and } \mathsf{C} \cap \mathbf{loc}(\Sigma_O) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

This function is extended to sequences of typed names in the obvious manner:

$$\mathsf{lnk}(n,\tilde{n}:\mathtt{T},\tilde{\mathtt{T}},\Sigma) \;=\; \mathsf{lnk}(n:\mathtt{T},\Sigma), \mathsf{lnk}(\tilde{n}:\tilde{\mathtt{T}},\Sigma')$$

where Σ' denotes $\Sigma + n : \mathtt{T}$. ∎

These revised actions give rise to a new (weak) bisimulation equivalence over configurations, $\approx$, defined in the usual way, but based on *derived actions*. We use

$$\Sigma \models M \approx N$$

to mean that the configurations $\Sigma \triangleright M$ and $\Sigma \triangleright N$ are bisimilar.

Example 3.4.3. Here we re-examine the systems in Example 3.3.8 and Example 3.3.9. We recall that in Example 3.3.8 we had the following actions with respect to the original lts: -

$$\Sigma_l \triangleright M_1 \xrightarrow{\mu_1} \Sigma + k_2 : \emptyset + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$
$$\Sigma_l \triangleright M_2 \xrightarrow{\mu_2} \Sigma + k_2 : \emptyset + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$

But Σ_l contains only one accessible node l; extending it with the new node k_2, linked to nothing does not increase the set of accessible nodes. Furthermore, increasing it with a new node k_3, linked to the inaccessible k_2 (in the case of $\Sigma \triangleright M_1$) also leads to no increase in the accessible nodes. Correspondingly, the calculations of $\mathsf{lnk}(k_2 : \emptyset, \Sigma)$ and $\mathsf{lnk}(k_3 : \{k_2\}, (\Sigma + k_2 : \emptyset))$ both lead to the empty link set.

Formally, we get the derived action

$$\Sigma \triangleright M_1 \overset{\alpha}{\longmapsto} \Sigma + k_2 : \emptyset + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$

where α is $(k_2 : \emptyset,\ k_3 : \emptyset)l : a!\langle k_2, k_3\rangle$. Similar calculations gives exactly the same derived action from M_2:

$$\Sigma \triangleright M_2 \overset{\alpha}{\longmapsto} \Sigma + k_2 : \emptyset + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\})\, l[\![P]\!]$$

Furthermore, if P contains no occurrence of k_1, we can go on to show

$$\Sigma + k_2 : \emptyset + k_3 : \{k_2\} \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \;\approx\; \Sigma + k_2 : \emptyset + k_3 : \emptyset \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!]$$

On the other hand, if P is $a!\langle k_1\rangle$, the subsequent transitions are different:

$$((\Sigma + k_2 : \emptyset) + k_3 : \{k_2\}) \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \overset{\beta_1}{\longmapsto} \ldots$$
$$((\Sigma + k_2 : \emptyset) + k_3 : \emptyset) \triangleright (\nu\, k_1 : \{l, k_2, k_3\}) l[\![P]\!] \overset{\beta_2}{\longmapsto} \ldots$$

where

$$\beta_1 \text{ is } (k_1 : \{k_1 \leftrightarrow k_2, k_1 \leftrightarrow k_3, k_2 \leftrightarrow k_3\}) l : a!\langle k_1\rangle$$
$$\beta_2 \text{ is } (k_1 : \{k_1 \leftrightarrow k_2, k_1 \leftrightarrow k_3\}) l : a!\langle k_1\rangle$$

We note that the link type associated with β_1 includes the additional component $\{k_2 \leftrightarrow k_3\}$, that was previously hidden, but is now made accessible as a result of scope extruding k_1; β_2 on the other hand, does not have this information in its link type. Based on this discrepancy between β_1 and β_2 we have

$$\Sigma_l \models M_1 \not\approx M_2$$

Revisiting Example 3.3.9, the three different actions of $\Sigma_l \triangleright N_1$, $\Sigma_l \triangleright N_2$ and $\Sigma_l \triangleright N_3$ now abstract to the same action $\Sigma_l \triangleright N_i \overset{\alpha}{\longmapsto} \ldots \triangleright l[\![\mathbf{0}]\!]$ for $i = 1..3$ where α is the label $(k : \emptyset)l : a!\langle k\rangle$. Thus we have

$$\Sigma_l \models N_i \approx N_j \quad \text{where } i, j = 1..3$$

as required. ■

3.5 Full-Abstraction

The purpose of this section is to show that our revised bisimulation equivalence is the correct one, in the sense that it coincides with some contextual equivalence appropriate to DπF; we need a generalisation of Theorem 2.3.5. This means developing a notion of reduction barbed congruence for DπF, similar to that in Definition 2.2.6 for DπLoc.

3.5.1 Reduction barbed congruence for DπF

The key to the definition is the isolation of the externally observable information in an extended environment. We use $\mathcal{I}$ to range over *knowledge representations*, pairs $\langle \mathcal{N}, \mathcal{O}\rangle$ where

- $\mathcal{N}$ is a set of names, as usual divided into $\mathbf{loc}(\mathcal{N})$ and $\mathbf{chan}(\mathcal{N})$,
- $\mathcal{O}$ is a linkset over $\mathcal{N}$.

These can be obtained from effective networks in the obvious manner:

$$\mathcal{I}(\Sigma) \stackrel{\text{def}}{=} \langle \Sigma_{\mathcal{N}}, \Sigma_{\mathcal{O}}\rangle$$

The key property of this subset of the information in a network representation is that it is preserved by derived actions:

Definition 3.5.1 (Action residuals). The partial function after ranges over knowledge representations $\mathcal{I}$ and external derived actions μ and returns knowledge representation, defined as:

- $\mathcal{I}$ after $(\tilde{n} : \tilde{\mathrm{L}})l : a!\langle V\rangle$ is defined as $\mathcal{I} + \tilde{n} : \tilde{\mathrm{L}}$
- $\mathcal{I}$ after $(\tilde{n} : \tilde{\mathrm{L}})l : a?(V)$ is defined as $\mathcal{I} + \tilde{n} : \tilde{\mathrm{L}}$
- $\mathcal{I}$ after kill : l is defined as $\mathcal{I} - l$

- $\mathcal{I}$ after $l \not\leftrightarrow k$ is defined as $\mathcal{I} - l \leftrightarrow k$ ■

Proposition 3.5.2. If $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$ where μ is a derived external action, then $\mathcal{I}(\Sigma')$ coincides with $\mathcal{I}(\Sigma)$ after μ

Proof. A straightforward induction on the inference of $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$. □

We find appropriate to use $\mathcal{I}$ as a means to define *the right circumstances*, necessary to allow an action μ to happen. Thus, we define the following predicate.

Definition 3.5.3 (Action Conditions). The predicate allows is defined over knowledge representations and action labels as:

- $\mathcal{I}$ allows τ is always true
- $\mathcal{I}$ allows $(\tilde{n} : \tilde{\mathsf{L}})l\!:\!a!\langle V\rangle$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}$
- $\mathcal{I}$ allows $(\tilde{n} : \tilde{\mathsf{L}})l : a?(V)$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}$, $V \subseteq \mathcal{I}_N$ and $\mathbf{dom}(\tilde{\mathsf{L}}) \subseteq (\mathbf{dom}(\mathcal{I}_O) \cup \tilde{n})$
- $\mathcal{I}$ allows kill$:l$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}$
- $\mathcal{I}$ allows $l \not\leftrightarrow k$ is true whenever $\mathcal{I} \vdash l : \mathbf{alive}, k : \mathbf{alive}$ ■

The following proposition states that the definition of allows is adequate with respect to our derived actions.

Proposition 3.5.4 (Adequacy of Allows). If $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$ then $\mathcal{I}(\Sigma)$ allows μ.

Proof. By induction on the derivation of $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$. □

We next establish that external actions of a configuration $\Pi \triangleright N$ are a function of the system N and the knowledge representation $\mathcal{I}(\Sigma)$. Before we prove this, we prove a lemma relating actions with the structure of the systems.

Lemma 3.5.5 (Derived Actions and Systems).

- if $\Sigma \triangleright N \overset{(\tilde{n}:\tilde{\mathsf{L}})l:a!\langle V\rangle}{\longmapsto} \Sigma + \tilde{n}\!:\!\tilde{\mathsf{T}} \triangleright N'$ where $\tilde{\mathsf{L}} = \mathsf{lnk}(\tilde{n}\!:\!\tilde{\mathsf{T}}, \Sigma)$ then
  - $N \equiv (\nu\, \tilde{n}\!:\!\tilde{\mathsf{T}})(\nu\, \tilde{m}\!:\!\tilde{\mathsf{U}})M|l[\![a!\langle V\rangle.P]\!]$
  - $N' \equiv (\nu\, \tilde{m}\!:\!\tilde{\mathsf{U}})M|l[\![P]\!]$
- if $\Sigma \triangleright N \overset{(\tilde{n}:\tilde{\mathsf{L}})l:a?(V)}{\longmapsto} \Sigma + \tilde{n}\!:\!\tilde{\mathsf{T}} \triangleright N'$ where $\tilde{\mathsf{L}} = \mathsf{lnk}(\tilde{n}\!:\!\tilde{\mathsf{T}}, \Sigma)$ then
  - $N \equiv (\nu\, \tilde{m}\!:\!\tilde{\mathsf{U}})M|l[\![a?(X).P]\!]$
  - $N' \equiv (\nu\, \tilde{m}\!:\!\tilde{\mathsf{U}})M|l[\![P\{{}^{V}\!/\!_{X}\}]\!]$
- if $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ where $\Sigma \vdash l : \mathbf{alive}$ and $\Sigma' \nvdash l : \mathbf{alive}$ then
  - $N \equiv N'|l[\![\mathsf{kill}]\!]$
- if $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ where $\Sigma \vdash l \leftrightarrow k$ and $\Sigma' \nvdash l \leftrightarrow k$ then

– $N \equiv N'|l[\![\text{break } k]\!]$ or $N \equiv N'|k[\![\text{break } l]\!]$

Proof. A straightforward induction on the inference of $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{L})l:a!\langle V\rangle} \Sigma' \triangleright N'$, $\Sigma \triangleright N \xrightarrow{(\tilde{n}:\tilde{L})l:a?(V)} \Sigma' \triangleright N'$ and $\Sigma \triangleright N \xrightarrow{\tau} \Sigma' \triangleright N'$. □

Proposition 3.5.6. If $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$ where μ is an external action, and $\mathcal{I}(\Sigma'')$ allows μ for some Σ'', then $\Sigma'' \triangleright N \overset{\mu}{\longmapsto} \Sigma''' \triangleright N'$ for some Σ'''

Proof. By induction on the inference of $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$, using Lemma 3.5.5 to infer the structure of N from μ. □

These lemmas indicate that these knowledge representations are the appropriate level of abstractions at which to generalise Definitions 2.2.3, 2.2.5 and 2.2.6 to DπF.

Definition 3.5.7 (Typed Relations for DπF). A *typed relation* over extended configurations is a binary relation between such configurations with the property that

$$\Sigma \triangleright M \;\mathcal{R}\; \Sigma' \triangleright N \;\text{ implies }\; \mathcal{I}(\Sigma) = \mathcal{I}(\Sigma')$$

We can mimic the notation in Definition 2.2.3 by writing

$$\mathcal{I} \models \Sigma \triangleright M \;\mathcal{R}\; \Sigma' \triangleright N$$

to mean that systems $\Sigma \triangleright M$ and $\Sigma' \triangleright N$ are related by $\mathcal{R}$ and that both $\mathcal{I}(\Sigma)$ and $\mathcal{I}(\Sigma')$ coincide with $\mathcal{I}$. ■

The definition of contextuality depends on what a given $\mathcal{I}$ allows to be observable; for this we adapt Definition 2.2.4.

Definition 3.5.8 (Observables). For any $\mathcal{I}$ let:

- $\mathcal{I} \vdash l{:}\mathbf{alive}$, if l is in $\mathbf{dom}(\mathcal{I}_O)$; this implies that l is not only alive, but in the accessible part of any Σ such that $\mathcal{I}(\Sigma)$ coincides with $\mathcal{I}$.
- $\mathcal{I} \vdash l \leftrightarrow k$, if $l \leftrightarrow k \in \mathcal{I}_O$; this implies that the link $l \leftrightarrow k$ is not only alive, but in the accessible part of any Σ such that $\mathcal{I}(\Sigma)$ coincides with $\mathcal{I}$.
- $\mathcal{I} \vdash \mathtt{T}$ if $\mathtt{T}$ is either $\mathtt{ch}$ or $\mathtt{loc[a,C]}$ such that $\mathtt{C} \subseteq \mathbf{dom}(\mathcal{I}_O)$; this means that such location types are linked only to observable locations.

We can now define the relation $\mathcal{I} \vdash O$ as:

- $\mathcal{I} \vdash l[\![P]\!]$ if $\mathbf{fn}(P) \subseteq \mathcal{I}_N$ and $\mathcal{I} \vdash l : \mathbf{alive}$
- $\mathcal{I} \vdash (\nu\, n{:}\mathtt{T})N$ if $\mathcal{I} \vdash \mathtt{T}$ and $(\mathcal{I} + n{:}\mathtt{T}) \vdash_{\mathsf{obs}} N$
- $\mathcal{I} \vdash M\,|\,N$ if $\mathcal{I} \vdash M$ and $\mathcal{I} \vdash N$

We can now adapt the notation of Definition 2.2.4 as:

$$\begin{aligned}
\Delta \vdash_{\mathsf{obs}} l{:}\mathbf{alive},\; l\leftrightarrow k,\; \mathtt{T},\; O &\stackrel{\text{def}}{=} \mathcal{I}(\Sigma(\Delta)) \vdash l{:}\mathbf{alive},\; l\leftrightarrow k,\; \mathtt{T},\; O\\
\Sigma \vdash_{\mathsf{obs}} l{:}\mathbf{alive},\; l\leftrightarrow k,\; \mathtt{T},\; O &\stackrel{\text{def}}{=} \mathcal{I}(\Sigma) \vdash l{:}\mathbf{alive},\; l\leftrightarrow k,\; \mathtt{T},\; O
\end{aligned}$$

The intuition of $\Delta \vdash_{\mathsf{obs}} O$ and $\Sigma \vdash_{\mathsf{obs}} O$ are still the same as that of Definition 2.2.4: an observer O is restricted to the observable network. However, the updated definition reflects the fact that the observable network is now not only defined in terms of live nodes but live, *reachable* nodes. ■

As a result of this adaptation, we can carry forward to this section the definition of contextual typed relations, defined earlier in 2.2.5. However, before we go on and define reduction barbed congruence for DπF terms, we need also to update the notion of a barb; a barb is observable by the context in DπF, if the location at which the barb occurs is alive *and observable*.

Definition 3.5.9. $\Sigma \triangleright N \Downarrow_{a@l}$ denotes an *observable barb* exhibited by the configuration $\Sigma \triangleright N$, on channel a at location l. Formally, it means that $\Delta(\Sigma) \triangleright N \longrightarrow^{*} \Delta(\Sigma') \triangleright N'$ for some $\Sigma' \triangleright N'$ such that $N' \equiv M | l[\![a!\langle V \rangle.Q]\!]$ and $\mathcal{I}(\Sigma) \vdash_{\mathsf{obs}} l : \textbf{alive}$. ■

With these modifications, Definition 2.2.6 can be applied to obtain a definition of *reduction barbed congruence* for DπF, which we denote by

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_2 \triangleright M_2 \;\;\text{whenever}\;\; \mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2)$$

Note that this enables us to compare aribtrary configurations, $\Sigma_1 \triangleright M_1$ and $\Sigma_2 \triangleright M_2$, but it can be specialised to simply comparing systems running on the same network. Let us write

$$\Sigma \models M \cong N$$

to mean that $\mathcal{I}(\Sigma) \models \Sigma \triangleright M \cong \Sigma \triangleright N$. Then, for example, the informal notation (3.1) used in Section 3.2.1 can be taken to mean

$$\Sigma(\Delta) \vdash M \cong N$$

The first main result of the chapter can now be stated:

Theorem 3.5.10. Suppose $\mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2) = \mathcal{I}$, for any effective configurations $\Sigma_1 \triangleright M_1$, $\Sigma_2 \triangleright M_2$ in DπF. Then:

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_2 \triangleright M_2 \;\;\text{if and only if}\;\; \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$$

This general result can also be specialised to the notation for comparing systems relative to a given network:

Corollary 3.5.11. In DπF, $\Sigma \models N \cong M$ if and only if $\Sigma \models N \;\approx\; M$. ■

The proof of the general theorem , which is quite complex, is detailed in the following two sections. The first section outlines the proof for *soundness*, that is, the adequacy of the derived action bisimulation as a means show that two configurations are reduction barbed congruent:

$$\Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2 \;\;\text{implies}\;\; \mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_1 \triangleright M_1$$

The second section outlines the proof for *completness*, that is, for any two configurations that are reduction barbed congruent, we can give a derived action bisimulation to show this:

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\cong\; \Sigma_1 \triangleright M_1 \;\text{ implies }\; \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$$

3.5.2 Soundness

The main task in proving that derived action bisimulation is sound is showing that $\approx$ is contextual. In addition to this, we also need to prove some preliminary results relating our lts with the reduction semantics of DπF.

We start by proving that the derived lts is *closed* over well formed effective configurations. We prove this with the aid of the following lemma, stating that there is also a special relationship between silent actions and residual networks.

Lemma 3.5.12. Internal transitions do not change the state of the network, unless a kill or a break l process in the configuration itself is consumed. Stated otherwise, if $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ then Σ' is either:-

1. Σ
2. $\Sigma - l$
3. $\Sigma - l \leftrightarrow k$

Proof. A straightforward induction on the inference of $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$. □

Proposition 3.5.13 (Closure). The derived lts given in Definition 3.4.1 forms a binary relation between well-defined effective configurations. Stated otherwise, if $\Sigma \vdash N$ and $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$ then $\Sigma' \vdash N'$.

Proof. By case analysis on the form of μ. We use Proposition 3.5.2 when μ is an external action and Lemma A.4.1 in sub-cases where we need to show that $\Sigma + n:\mathtt{T}$ is still a valid effective network. When μ is an internal action, $\mu = \tau$, we use Lemma 3.5.12. □

The next important sanity check for our lts is that our formulation of *internal activity*, namely $\overset{\tau}{\longmapsto}$, is in agreement, in some sense, with the reduction semantics.

Proposition 3.5.14 (Reductions correspond to τ-actions).

- $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$ implies $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N''$ for some $N'' \equiv N'$
- $\Sigma \triangleright N \overset{\tau}{\longmapsto} \Sigma' \triangleright N'$ implies $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$

Proof. The proof for the first clause is by induction on why $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$. The proof for the second clause is also by induction. Since the internal transition rule (l-par-comm) is defined in terms of input and output actions, we make use of Lemma 3.5.5 in our induction. □

We now embark on the main task of this section, that of showing that our bisimulation, $\approx$, is contextual. This proof relies heavily on the Composition and Decomposition Lemmas stated below, explaining how actions can be composed of, or decomposed into, other actions. Both Composition and Decomposition Lemmas make use of the following (specific) lemma, which is a slight variation on Proposition 3.5.6; we note that we could not have used Proposition 3.5.6 in this case because the type of the bound input action changes as shown below.

Lemma 3.5.15 (Input actions and the maximal observer view).

- If $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{K}})l:a?(V)} \Sigma' \triangleright N'$ then $\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{L}})l:a?(V)} \Sigma'' \triangleright N'$ where $\tilde{\mathsf{K}} = \tilde{\mathsf{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$.
- If $\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{L}})l:a?(V)} \Sigma' \triangleright N'$ and $\mathcal{I}(\Sigma) \vdash l:\mathbf{alive}$ then $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{K}})l:a?(V)} \Sigma'' \triangleright N'$ where $\tilde{\mathsf{K}} = \tilde{\mathsf{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$.

Proof. The proof uses Lemma 3.5.5 to infer the structure of N and the progresses by induction on the structure of N, similar to the proof for Proposition 3.5.6. □

Lemma 3.5.16 (Composition).

- Suppose $\Sigma \triangleright M \xmapsto{\mu} \Sigma' \triangleright M'$. If $\Sigma \vdash N$ for arbitrary system N, then $\Sigma \triangleright M|N \xmapsto{\mu} \Sigma' \triangleright M'|N$ and $\Sigma \triangleright N|M \xmapsto{\mu} \Sigma \triangleright N|M$.
- Suppose $\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathsf{L}})l:a!\langle V\rangle} \Sigma' \triangleright M'$ and $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{K}})l:a?(V)} \Sigma'' \triangleright N'$ where $\tilde{\mathsf{K}} = \tilde{\mathsf{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$. Then
  - $\Sigma \triangleright M|N \xmapsto{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathsf{T}})M'|N'$ where $\tilde{\mathsf{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathsf{T}}, \Sigma)$
  - $\Sigma \triangleright N|M \xmapsto{\tau} \Sigma \triangleright (\nu\, \tilde{n}:\tilde{\mathsf{T}})N'|M'$ where $\tilde{\mathsf{L}} = \mathsf{lnk}(\tilde{n}:\tilde{\mathsf{T}}, \Sigma)$

Proof. The proof for the first clause is trivial, by using (l-par-ctxt). We here outline the proof for the second clause. From the hypothesis

$$\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathsf{L}})l:a!\langle V\rangle} \Sigma + \tilde{n}:\tilde{\mathsf{T}} \triangleright M' \tag{3.5}$$

and Proposition 3.5.4 we know $\mathcal{I}(\Sigma)$ allows $(\tilde{n} : \tilde{\mathsf{L}})l : a!\langle V\rangle$ and thus $\mathcal{I}(\Sigma) \vdash l:\mathbf{alive}$. It is obvious that $\mathcal{I}(\uparrow(\Sigma)) \vdash l:\mathbf{alive}$ as well and hence

$$\mathcal{I}(\uparrow(\Sigma)) \text{ allows } (\tilde{n} : \tilde{\mathsf{L}})l : a!\langle V\rangle \tag{3.6}$$

From (3.5), (3.6) and Proposition 3.5.6 we derive

$$\uparrow(\Sigma) \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathsf{L}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathsf{T}} \triangleright M'$$

and from (l-deriv-2) we conclude

$$\uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathsf{T}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathsf{T}} \triangleright M' \tag{3.7}$$

From the hypotheses $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{K}})l:a?(V)} \Sigma'' \triangleright N'$ and $\tilde{\mathsf{K}} = \tilde{\mathsf{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$ and Lemma 3.5.15 we immediately derive

$$\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathsf{L}})l:a?(V)} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathsf{T}} \triangleright N'$$

and by (l-deriv-3) we derive

$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright N' \tag{3.8}$$

Hence, by (3.7), (3.8), (l-par-comm) and (l-deriv-1) we conclude

$$\begin{aligned} \Sigma \triangleright M|N &\overset{\tau}{\longmapsto} \Sigma \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{T}})M'|N' \\ \Sigma \triangleright N|M &\overset{\tau}{\longmapsto} \Sigma \triangleright (\nu\,\tilde{n}:\tilde{\mathrm{T}})N'|M' \end{aligned}$$

as required. □

Lemma 3.5.17 (Decomposition). If $\Sigma \triangleright M|N \overset{\mu}{\longmapsto} \Sigma' \triangleright M'$ where $\Sigma \vdash_{\mathsf{obs}} M$ or $\Sigma \vdash_{\mathsf{obs}} N$ then one of the following conditions hold:

1. M' is $M''|N$, where $\Sigma \triangleright M \overset{\mu}{\longmapsto} \Sigma' \triangleright M''$.
2. M' is $M|N'$ and $\Sigma \triangleright N \overset{\mu}{\longmapsto} \Sigma' \triangleright N'$.
3. M' is $(\nu\,\tilde{n}:\tilde{\mathrm{T}})M''|N'$, μ is τ, $\Sigma' = \Sigma$ and either
   - $\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{L}})l:a!\langle V\rangle} \Sigma'' \triangleright M''$ and $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathrm{K}})l:a?(V)} \Sigma''' \triangleright N'$
   - $\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{K}})l:a?(V)} \Sigma'' \triangleright M''$ and $\Sigma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathrm{L}})l:a!\langle V\rangle} \Sigma''' \triangleright N'$

   where $\tilde{\mathrm{K}} = \tilde{\mathrm{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$

Proof. The proof progressed by induction on the derivation of $\Sigma \triangleright M|N \overset{\mu}{\longmapsto} \Sigma' \triangleright M'$. We focus on the case where $\mu = \tau$, and the last two rules used in our derivation were (l-deriv-1) and (l-par-comm). From the inductive hypothesis of (l-par-comm) we derive

$$\Sigma' = \Sigma \tag{3.9}$$

$$M' \text{ is } (\nu\,\tilde{n}:\tilde{\mathrm{T}})M'|N' \tag{3.10}$$

$$\uparrow(\Sigma) \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright M' \tag{3.11}$$

$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright N' \tag{3.12}$$

or viceversa. From (3.11), (3.12), (l-deriv-2) and (l-deriv-3) we get

$$\uparrow(\Sigma) \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{L}})l:a!\langle V\rangle} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright M' \tag{3.13}$$

$$\uparrow(\Sigma) \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathrm{L}})l:a?(V)} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathrm{T}} \triangleright N' \tag{3.14}$$

From the assumption that $\Sigma \vdash_{\mathsf{obs}} M$ or $\Sigma \vdash_{\mathsf{obs}} N$ we derive $\Sigma \vdash_{\mathsf{obs}} l:\mathbf{alive}$ meaning

$$\mathcal{I}(\Sigma) \vdash l:\mathbf{alive} \tag{3.15}$$

From (3.15) we derive

$$\mathcal{I}(\Sigma) \text{ allows } (\tilde{n}:\tilde{\mathrm{T}})l : a!\langle V\rangle \tag{3.16}$$

and by (3.13), (3.16) and Proposition 3.5.6 we deduce

$$\Sigma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{L}})l:a!\langle V\rangle} \Sigma'' \triangleright M'$$

Moreover, by (3.14), (3.15) and Lemma 3.5.15 we deduce

$$\uparrow(\Sigma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \uparrow(\Sigma) + \tilde{n}:\tilde{\mathtt{T}} \triangleright N'$$

where $\tilde{\mathtt{K}} = \tilde{\mathtt{L}}/\mathbf{dom}(\Sigma_{\mathcal{H}})$ as required. □

We now turn our attention to the actual proof for the main proposition of this section, namely that bisimulation, $\approx$, is contextual. We prove this by inductively defining the largest contextual relation whose base element are bisimilar configurations and then show its closure with respect to our derived actions. Based on such a proof, we still require three (specific) lemmas to help us stitch up this proof and guarantee closure. The first lemma is prompted by the first two conditions of the Decomposition Lemma 3.5.17, namely that observing code may alter the state of the network by inducing failure. We thus need the following lemma to guarantee closure.

Lemma 3.5.18 (Strong Failure versus Weak Failure). Suppose $\Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$. Then there exists some M_2', M_2'' such that:

- $\Sigma_2 \triangleright M_2 \overset{\widehat{\tau}}{\Longmapsto} \Sigma_2 \triangleright M_2'$ and $(\Sigma_2 - l) \triangleright M_2' \overset{\tau}{\Longmapsto} (\Sigma_2 - l) \triangleright M_2''$ such that $(\Sigma_1 - l) \triangleright M_1 \approx (\Sigma_2 - l) \triangleright M_2''$
- $\Sigma_2 \triangleright M_2 \overset{\widehat{\tau}}{\Longmapsto} \Sigma_2 \triangleright M_2'$ and $(\Sigma_2 - l \leftrightarrow k) \triangleright M_2' \overset{\tau}{\Longmapsto} (\Sigma_2 - l \leftrightarrow k) \triangleright M_2''$ such that $(\Sigma_1 - l \leftrightarrow k) \triangleright M_1 \approx (\Sigma_2 - l \leftrightarrow k) \triangleright M_2''$

Proof. We here prove the first clause and leave the second similar clause for the interested reader. If $\Sigma_1 \nvdash l : \mathbf{alive}$ then $\Sigma_1 - l$ is simply Σ_1 and the result is trivial. Otherwise $\Sigma_1 \triangleright M_1 \overset{\mathsf{kill}:l}{\longmapsto} \Sigma_1 - l \triangleright M_1$ and hence $\Sigma_2 \triangleright M_2 \overset{\mathsf{kill}:l}{\Longmapsto} \Sigma_2 - l \triangleright M''$ for some $\Sigma_2 - l \triangleright M''$ such that $\Sigma_1 - l \triangleright M_1 \approx \Sigma_2 - l \triangleright M''$. By expanding our the derivation $\Sigma_2 \triangleright M \overset{\mathsf{kill}:l}{\Longmapsto} (\Sigma_2 - l) \triangleright M''$ we get the required missing M' to complete the proof. □

The next two proofs concern observing code. In the definition of contextual relations, we validate observer code, O, with respect to the external view of a network Σ, that is $\Sigma \vdash_{\mathsf{obs}} O$. We here prove that such a relationship is preserved by actions and network extensions that may involve revealing more hidden components to the observer.

Lemma 3.5.19 (Observers and Actions). If $\Sigma \vdash_{\mathsf{obs}} O$ and $\Sigma \triangleright O \overset{\mu}{\longmapsto} \Sigma \ \mathsf{after}\ \mu \triangleright O'$ then $(\Sigma \ \mathsf{after}\ \mu) \vdash_{\mathsf{obs}} O'$.

Proof. The proof is similar to that of Proposition3.5.13. We use Lemma 3.5.5 to infer the structure of O, O' from μ and Lemma 3.5.2 to infer the structure of $\Sigma \ \mathsf{after}\ \mu$ and then show that $(\Sigma \ \mathsf{after}\ \mu) \vdash_{\mathsf{obs}} O'$. □

Lemma 3.5.20 (Observers and Network extensions). If $\Sigma + n:\mathtt{U} \vdash_{\mathsf{obs}} O$ where $\Sigma \vdash_{\mathsf{obs}} \mathtt{U}$, that is n is only linked to locations in the observable part of Σ and thus no hidden state is revealed as a result of the extension, then $\Sigma + n:\mathtt{T} \vdash_{\mathsf{obs}} O$ for any $\mathtt{T}$ where $\mathtt{U} = \mathtt{T}/\mathbf{dom}(\Sigma_{\mathcal{H}})$.

Proof. The proof progresses by a simple induction on the structure of O. □

We are finally in a position to prove that our bisimulation, $\approx$, is a contextual relation, according to Definition 2.2.5.

Proposition 3.5.21 (Contextuality of Behavioural Equivalence). If two configurations are bisimilar, they are also bisimilar under any context. Stated otherwise, $\mathcal{I} \models \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2$ implies that for $\mathcal{I} \vdash O, \mathtt{T}$ and n fresh in $\mathcal{I}$ we have:

- $\mathcal{I} \models \Sigma_1 \triangleright M_1|O \approx \Sigma_2 \triangleright M_2|O$ and $\mathcal{I} \models \Sigma_1 \triangleright O|M_1 \approx \Sigma_2 \triangleright O|M_2$
- $\mathcal{I}{+}n{:}\mathtt{T} \models \Sigma_1{+}n{:}\mathtt{T} \triangleright M_1 \approx \Sigma_2{+}n{:}\mathtt{T} \triangleright M_2$

Proof. The proof progresses by the inductive definition a relation $\mathcal{R}$ as the largest typed relation over configurations satisfying:

$$\mathcal{R} = \left\{ \begin{array}{ll} \langle \Sigma_1 \triangleright M_1,\ \Sigma_2 \triangleright M_2 \rangle & \mid \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2 \\[2ex] \begin{array}{l}\langle \Sigma_1 \triangleright M_1|O,\ \Sigma_2 \triangleright M_2|O \rangle \\ \langle \Sigma_1 \triangleright O|M_1,\ \Sigma_2 \triangleright O|M_2 \rangle\end{array} & \left|\ \Sigma_1 \triangleright M_1 \mathcal{R} \Sigma_2 \triangleright M_2 \right. \\[2ex] \langle \Sigma_1{+}n{:}\mathtt{T} \triangleright M_1|O,\ \Sigma_2{+}n{:}\mathtt{T} \triangleright M_2|O \rangle & \left|\ \begin{array}{l}\mathcal{I} \models \Sigma_1 \triangleright M_1 \ \mathcal{R}\ \Sigma_2 \triangleright M_2, \\ \mathcal{I} \vdash \mathtt{T} \text{ and } n \text{ is fresh}\end{array}\right. \\[2ex] \langle \Sigma_1 \triangleright (\nu\, n{:}\mathtt{T})M_1,\ \Sigma_2 \triangleright (\nu\, n{:}\mathtt{U})M_2 \rangle & \mid \Sigma_1{+}n{:}\mathtt{T} \triangleright M_1 \ \mathcal{R}\ \Sigma_2{+}n{:}\mathtt{U} \triangleright M_2 \end{array} \right\}$$

and showing that $\mathcal{R} \subseteq \approx$; since $\approx$ is the biggest possible relation, this would mean that it is contextual. We note that our definition of contextual relations, Definition 2.2.5, would amount to a special case of the contexts defined for $\mathcal{R}$ because it is only defined in terms of the second and third cases of the relation $\mathcal{R}$, namely contexts involving more systems in parallel and contexts involving a bigger network. The fourth and last context case, that of name scoping, is required to ensure the closure of $\mathcal{R}$. All this is fairly standard with the exception that the type at which names are scoped in the fourth case, that is T and U, may not be the same because of the potentially different hidden states in Σ_1 and Σ_2.

Before we delve into the actual proof we also note that Lemma 3.5.18 can be easily extended from $\approx$ to $\mathcal{R}$ as:

Lemma 3.5.22. If $\Sigma_1 \triangleright M_1 \,\mathcal{R}\, \Sigma_2 \triangleright M_2$, then there exist some M_2', M_2'' such that:

- $\Sigma_2 \triangleright M_2 \stackrel{\widehat{\tau}}{\Longrightarrow} \Sigma_2 \triangleright M_2'$ and $\Sigma_2 - l \triangleright M_2' \stackrel{\tau}{\Longrightarrow} \Sigma_2 - l \triangleright M_2''$, where $\Sigma_1 - l \triangleright M_1 \,\mathcal{R}\, \Sigma_2 - l \triangleright M_2''$
- $\Sigma_2 \triangleright M_2 \stackrel{\widehat{\tau}}{\Longrightarrow} \Sigma_2 \triangleright M_2'$ and $(\Sigma_2 - l \leftrightarrow k) \triangleright M_2' \stackrel{\tau}{\Longrightarrow} (\Sigma_2 - l \leftrightarrow k) \triangleright M_2''$, where $(\Sigma_1 - l \leftrightarrow k) \triangleright M_1 \,\mathcal{R}\, (\Sigma_2 - l \leftrightarrow k) \triangleright M_2''$

The proof for the above is by induction on why $\Sigma_1 \triangleright M_1 \,\mathcal{R}\, \Sigma_2 \triangleright M_2$; the base case follows from Lemma 3.5.18 and the three inductive cases are straightforward.

To prove that $\mathcal{R}$ is a bisimulation, we take an arbitrary $\mathcal{I} \models \Sigma_1 \triangleright M_1 \ \mathcal{R}\ \Sigma_2 \triangleright M_2$ and any action $\Sigma_1 \triangleright M_1 \stackrel{\mu}{\longmapsto} \Sigma_1' \triangleright M_1'$; we then have to show that $\Sigma_2 \triangleright M_2$ can match this move

by performing a weak action $\Sigma_2 \triangleright M_2 \overset{\hat{\mu}}{\Longmapsto} \Sigma_2' \triangleright M_2'$ such that $\mathcal{I}' \models \Sigma_1' \triangleright M_1' \;\mathcal{R}\; \Sigma_2' \triangleright M_2'$. The proof progress by induction on why $\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\mathcal{R}\; \Sigma_2 \triangleright M_2$; The first case, that is if $\mathcal{I} \models \Sigma_1 \triangleright M_1 \;\approx\; \Sigma_2 \triangleright M_2$ is immediate; the remaining three cases require a bit more work. We here focus on the second case, where

$$\Sigma_1 \triangleright M_1|O \,\mathcal{R}\, \Sigma_2 \triangleright M_2|O \text{ because } \mathcal{I} \models \Sigma_1 \triangleright M_1 \;\mathcal{R}\; \Sigma_2 \triangleright M_2 \text{ and } \mathcal{I} \vdash O \tag{3.17}$$

which is also the most involving and leave the remaining two cases for the interested reader.

We thus assume $\Sigma_1 \triangleright M_1|O \overset{\mu}{\longmapsto} \Sigma_1' \triangleright M_1'$. We decompose this action using the Decomposition Lemma 3.5.17 and focus on the most difficult case, where

$$M_1' \text{ is } (\nu\,\tilde{n}:\tilde{\mathtt{T}})M_1'|O',\ \mu \text{ is } \tau \text{ and } \Sigma_1' = \Sigma_1 \tag{3.18}$$

$$\Sigma_1 \triangleright M_1 \quad \xrightarrow{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle} \quad \Sigma_1 + \tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \tag{3.19}$$

$$\Sigma_1 \triangleright O \quad \xrightarrow{(\tilde{n}:\tilde{\mathtt{K}})l:a?(V)} \quad \Sigma_1 + \tilde{n}:\tilde{\mathtt{U}} \triangleright O' \text{ where } \tilde{\mathtt{U}} = \tilde{\mathtt{T}}/\mathbf{dom}(\Sigma_{1\mathcal{H}}) \tag{3.20}$$

From (3.17) and (3.19) we derive the matching weak action

$$\Sigma_2 \triangleright M_2 \overset{(\tilde{n}:\tilde{\mathtt{L}})l:a!\langle V\rangle}{\Longmapsto} \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \;\mathcal{R}\; \Sigma_1 + \tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \tag{3.21}$$

where we note the different types $\tilde{\mathtt{T}}$ and $\tilde{\mathtt{W}}$ at which the two networks Σ_1 and Σ_2 are updated; there are updates to the hidden part of the networks which we abstract away in the linktype $\tilde{\mathtt{L}}$. From (3.21) and the hypothesis of (l-deriv-2) we obtain

$$\Sigma_2 \triangleright M_2 \overset{(\tilde{n}:\tilde{\mathtt{W}})l:a!\langle V\rangle}{\Longrightarrow} \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'$$

which can be decomposed as

$$\Sigma_2 \triangleright M_2 \quad \Longrightarrow \quad \Sigma_2'' \triangleright M_2'' \tag{3.22}$$

$$\Sigma_2'' \triangleright M_2'' \quad \xrightarrow{(\tilde{n}:\tilde{\mathtt{W}})l:a!\langle V\rangle} \quad \Sigma_2'' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2''' \tag{3.23}$$

$$\Sigma_2'' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2''' \quad \Longrightarrow \quad \Sigma_2' + \tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \tag{3.24}$$

From (3.22), $\mathcal{I} \vdash O$ and (l-par-ctxt) we get

$$\Sigma_2 \triangleright M_2|O \quad \Longrightarrow \quad \Sigma_2'' \triangleright M_2''|O \tag{3.25}$$

From the fact that $\mathcal{I}(\Sigma_1) = \mathcal{I}(\Sigma_2)$ and $\mathcal{I}(\Sigma_1 + \tilde{n}:\tilde{\mathtt{T}}) = \mathcal{I}(\Sigma_2' + \tilde{n}:\tilde{\mathtt{W}})$ from (3.21) we know that the visible part of Σ_2'' and Σ_2' did not change as a result of the silent transitions in (3.22) and (3.24) and thus

$$\mathcal{I}(\Sigma_2'') \;=\; \mathcal{I}(\Sigma_2') \;=\; \mathcal{I}(\Sigma_2) \;=\; \mathcal{I}(\Sigma_1) \tag{3.26}$$

and by (3.26), (3.20) and Lemma 3.5.6 we get

$$\Sigma_2'' \triangleright O \xrightarrow{(\tilde{n}:\tilde{\mathtt{U}})l:a?(V)} \Sigma_2'' + \tilde{n}:\tilde{\mathtt{U}} \triangleright O' \text{ where } \tilde{\mathtt{U}} = \tilde{\mathtt{W}}/\mathbf{dom}(\Sigma_{2\,\mathcal{H}}'') \tag{3.27}$$

At this point we note that from (3.26) and (3.17) we derive

$$\Sigma_2'' \vdash_{\mathsf{obs}} O \tag{3.28}$$

and from (3.27), (3.28), Lemma 3.5.19 and Lemma 3.5.20 we obtain

$$\mathcal{I}(\Sigma_2''+\tilde{n}:\tilde{\mathtt{U}}) \vdash O' \text{ and } \mathcal{I}(\Sigma_2''+\tilde{n}:\tilde{\mathtt{W}}) \vdash O' \tag{3.29}$$

Combining the derived action of (3.23) using (l-deriv-2), the derived action of (3.27) using (l-deriv-3), (3.28), and the Composition Lemma 3.5.16, we obtain

$$\Sigma_2'' \triangleright M_2''|O \xrightarrow{\tau} \Sigma_2'' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'''|O' \tag{3.30}$$

From (3.24), (3.29) and (l-par-ctxt) we obtain

$$\Sigma_2''+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'''|O' \Longrightarrow \Sigma_2'+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'|O'$$

and by applying (l-rest) we get

$$\Sigma_2'' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'''|O' \Longrightarrow \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O' \tag{3.31}$$

and thus by combining (3.25), (3.30) and (3.31) and then applying (l-deriv-1) we obtain the matching move

$$\Sigma_2 \triangleright M_2|O \stackrel{\tau}{\Longmapsto} \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O' \tag{3.32}$$

The only thing remaining is to show that the two residuals are in $\mathcal{R}$, that is

$$\Sigma_1 \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M_1'|O' \;\mathcal{R}\; \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O'$$

From (3.21) we know

$$\mathcal{I}' \models \Sigma_1+\tilde{n}:\tilde{\mathtt{T}} \triangleright M_1' \quad \mathcal{R} \quad \Sigma_2'+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2' \tag{3.33}$$

and from (3.29) and (3.33) we deduce $\mathcal{I}' \vdash O'$ and thus from the definition of $\mathcal{R}$ we obtain

$$\mathcal{I}' \models \Sigma_1+\tilde{n}:\tilde{\mathtt{T}} \triangleright M_1'|O' \quad \mathcal{R} \quad \Sigma_2'+\tilde{n}:\tilde{\mathtt{W}} \triangleright M_2'|O'$$

and again from the last case of the definition of $\mathcal{R}$

$$\mathcal{I} \;\models\; \Sigma_1 \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M_1'|O' \quad \mathcal{R} \quad \Sigma_2' \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{W}})M_2'|O'$$

as required. □

We now conclude this section by showing that bisimulation is sound with respect to reduction barbed congruence.

Proposition 3.5.23 (Soundness).

$$\mathcal{I} \models \Sigma_1 \triangleright M_1 \approx \Sigma_2 \triangleright M_2 \text{ implies } \mathcal{I} \models \Sigma_1 \triangleright M_1 \cong \Sigma_2 \triangleright M_2$$

Proof. To prove the above statement, it is sufficient to check that $\approx$ satisfies the defining properties of $\cong$. It is obviously reduction closed, from the relationship between τ-actions and the reduction semantics given in Proposition 3.5.14. Barb preserving is also straightforward, from Proposition 3.5.5 and the direct relationship between barbs and output actions. Finally, Proposition 3.5.21 proves that $\approx$ is also contextual. □

3.5.3 Completness

In this section we prove that our bisimulation is also *complete* with respect to reduction barbed congruence. This entails showing that reduction barbed congruence is preserved by actions, based on the proof developed earlier in [HR04, HMR04]. At the heart of this proof, we show that the effect of each external action can be mimicked precisely by a specific context, a concept we refer to as *definability*.

We start this section by proving an obvious, though not explicit, property stating that reduction babrbed congruence is preserved by scoping.

Proposition 3.5.24 (Scoping and reduction barbed congruence). If two configurations are reduction barbed congruent, scoping a channel or location name on both sides would still yield two reduction barbed congruent configurations. Stated otherwise,

$$(\Sigma_M + n:\mathtt{T}) \triangleright M \cong (\Sigma_N + n:\mathtt{U}) \triangleright N \text{ implies } \Sigma_M \triangleright (\nu\, n:\mathtt{T})M \cong \Sigma_N \triangleright (\nu\, n:\mathtt{U})N$$

Proof. We define the relation $\mathcal{R}$ as:

$$\mathcal{R} \;=\; \left\{ \langle \Sigma_M \triangleright (\nu\, n:\mathtt{T})M,\ \Sigma_N \triangleright (\nu\, n:\mathtt{U})N\rangle \;\middle|\; (\Sigma_M + n:\mathtt{T}) \triangleright M \cong (\Sigma_N + n:\mathtt{U}) \triangleright N \right\}$$

and prove that $\mathcal{R}$ has the defining properties of $\cong$. It is clearly reduction closed using (r-ctxt-res); it is also easy to show it is barb preserving since $\Sigma_M \triangleright (\nu\, n:\mathtt{T})M \Downarrow_{a@l}$ implies $(\Sigma_M + n:\mathtt{T}) \triangleright M \Downarrow_{a@l}$. Finally, contextuality is also trivial. As an example, assume $\mathcal{I}(\Sigma_M) \vdash O$ and we have to show that

$$\Sigma_M \triangleright O\,|\,(\nu\, n:\mathtt{T})(M)\; \mathcal{R}\; \Sigma_N \triangleright O\,|\,(\nu\, n:\mathtt{U})N.$$

It is clear that $\Sigma_M + n:\mathtt{T} \vdash_{\mathsf{obs}} O$ and $\Sigma_N + n:\mathtt{U} \vdash_{\mathsf{obs}} O$ and thus by contextuality of $\cong$, we have $(\Sigma_M + n:\mathtt{T}) \triangleright O\,|\,M \cong (\Sigma_N + n:\mathtt{U}) \triangleright O\,|\,N$ from which the result follows. □

Our external actions can affect both the system part of our configuration as well as the network representation and the main differences between the definability proofs presented here and those in [HR04, HMR04] lie in the effects an action has on the network representation. In the following proofs, we model an action's effect on a network using two different kinds of new construct introduced in DπF; the first kind of constructs *induce* faults as changes in the network representation and these include kill and break l; the second kind *observe* the current state of the network and the only example is the ping $l.P\lceil Q\rceil$ construct. The first lemma we consider, establishes a relationship between the labels kill : l and $l \leftrightarrow k$ and the constructs inducing faults in the observable network representation; this proof is complicated by the asynchronous nature of the constructs kill and break l.

Lemma 3.5.25 (Inducing faults).

- Suppose $\Sigma \vdash_{\mathsf{obs}} l : \textbf{alive}$. Then:
  - $\Sigma \triangleright N \xmapsto{\mathsf{kill}:l} \Sigma' \triangleright N'$ implies $\Sigma \triangleright N|l[\![\mathsf{kill}]\!] \longrightarrow \Sigma' \triangleright N'$

  - $\Sigma \triangleright N|l[\![\mathsf{kill}]\!] \longrightarrow \Sigma' \triangleright N'$, where $\Sigma' \nvdash_{\mathsf{obs}} l:\textbf{alive}$ implies $\Sigma \triangleright N \overset{\mathsf{kill}:l}{\longmapsto} \Sigma' \triangleright N''$ such that $N' \equiv N''$

- Suppose $\Sigma \vdash_{\mathsf{obs}} l \leftrightarrow k$. Then:
  - $\Sigma \triangleright N \overset{l\not\leftrightarrow k}{\longmapsto} \Sigma' \triangleright N'$ implies $\Sigma \triangleright N|l[\![\mathsf{break}\, k]\!] \longrightarrow \Sigma' \triangleright N'$
  - $\Sigma \triangleright N|l[\![\mathsf{break}\, k]\!] \longrightarrow \Sigma' \triangleright N'$, where $\Sigma' \nvdash_{\mathsf{obs}} l \leftrightarrow k$ implies $\Sigma \triangleright N \overset{l\not\leftrightarrow k}{\longmapsto} \Sigma' \triangleright N''$ such that $N' \equiv N''$

Proof. The first clause for the action $\mathsf{kill} : l$ is proved by induction on the derivation $\Sigma \triangleright N \overset{\mathsf{kill}:l}{\longrightarrow} \Sigma' \triangleright N'$. The second clause uses induction on the structure of $\Sigma \triangleright N$, with a subsidiary induction on the derivation of $\Sigma \triangleright N|l[\![\mathsf{kill}]\!] \longrightarrow \Sigma'' \triangleright N''$. The proof for the two clauses of the action $l \not\leftrightarrow k$ is similar. □

In the next lemma we show that for any network Σ, the context can determine the exact state of the observable network Σ_O. We define the process $verStat_k^{\mathcal{I}}(x)$ that runs at a location k, which should be connected to all observable locations in a network Σ. It returns an output on the parameterised channel x if and only if $\mathcal{I}(\Sigma) = \mathcal{I}$. Its implementation is based on the state observing construct $\mathsf{ping}\, l.P\lceil Q \rceil$: the sub-process, $verObs_k^{\mathcal{I}}(x)$, first checks that all *inaccessible* locations in $\mathcal{I}$, expressed as $\mathcal{I}' + l:\emptyset$ below, are indeed inaccessible and then checks that the *accessible* locations, expressed as $\mathcal{I} + l:\mathtt{L}$ where $\mathtt{L} \neq \emptyset$, satisfy the state declared in $\mathcal{I}$, using the sub-process $verLoc_k(x, y_1, y_2, z)$. This last subprocess, goes to the parameterised location x and checks that all its live connections and dead connections correspond to y_1 and y_2 respectively, returning a output on channel z if it is the case. The following lemma formalises the intuition that when run at an appropriate location, $verStat_k^{\mathcal{I}}(x)$ does satisfy the intended behaviour.

$$verStat_k^{\mathcal{I}}(x) \;\Leftarrow\; (\nu\, sync)\begin{pmatrix} verObs_k^{\mathcal{I}}(sync) \\ \mid \underbrace{sync?().\ldots sync?()}_{|\mathbf{loc}(\mathcal{N})|}.x!\langle\rangle \end{pmatrix}$$

$$\begin{aligned}
verObs_k^{\langle\emptyset,\emptyset\rangle}(x) &\Leftarrow \mathbf{0} \\
verObs_k^{\mathcal{I}+n:\emptyset}(x) &\Leftarrow verObs_k^{\mathcal{I}}(x) \mid \mathsf{ping}\; l.\lceil y!\langle\rangle \rceil \\
verObs_k^{\mathcal{I}+l:\mathtt{L}}(x),\ \mathtt{L}\neq\emptyset &\Leftarrow verObs_k^{\mathcal{I}}(x) \mid verLoc_k(l, \mathbf{dom}(\mathtt{L}), \mathbf{loc}(\mathcal{I}_{\mathcal{N}})/\mathbf{dom}(\mathtt{L}), x)
\end{aligned}$$

$$verLoc_k(x, y_1, y_2, z) \;\Leftarrow\; (\nu\, sync)\mathsf{go}\; x.\begin{pmatrix} \prod_{l\in y_1} \mathsf{go}\; l.\mathsf{go}\; x.sync!\langle\rangle \\ \mid \prod_{l\in y_2} \mathsf{ping}\; l.\lceil sync!\langle\rangle \rceil \\ \mid \underbrace{sync?().\ldots sync?()}_{|\mathbf{loc}(\mathcal{I}_{\mathcal{N}})|}.\mathsf{go}\; k.z!\langle\rangle \end{pmatrix}$$

Lemma 3.5.26 (Observable Network). If for arbitrary network representation Σ:

$$\Sigma_+ = \Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\Sigma_O)] + \textsc{succ} : \mathtt{ch}$$

Then,

$$\Sigma_+ \triangleright k_0[\![verStat^{\mathcal{I}}_{k_0}(\textsc{succ})]\!] \longrightarrow^* \Sigma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] \text{ iff } \mathcal{I} = \mathcal{I}(\Sigma)$$

Proof. We prove this lemma by contradiction. We analyse all the possible cases why $\mathcal{I} \neq \mathcal{I}(\Sigma)$ and then show that for each of these cases,

$$\Sigma_+ \triangleright k_0[\![verStat^{\mathcal{I}}_{k_0}(\textsc{succ})]\!] \not\longrightarrow^* \Sigma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] \qquad \square$$

We are now in a position to prove definability for every external action in DπF. We use $\mathbf{bn}(\mu)$ to denote the bound names (and their corresponding types) in the action μ; note this is empty for all actions apart from bound input and bound output. In order to complete the proof, we also require the following lemma.

Lemma 3.5.27. If $n \notin \mathbf{fn}(N)$ and $\Sigma \vdash N$ then

- $\Sigma + n : \mathtt{T} \triangleright N \longrightarrow \Sigma' + n : \mathtt{T} \triangleright N'$ implies $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$
- $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$ implies $\Sigma + n : \mathtt{T} \triangleright N \longrightarrow \Sigma' + n : \mathtt{T} \triangleright N'$

Proof. The proofs are by induction on the structure of N for $\Sigma \vdash N$ and by induction on the derivations of $\Sigma \triangleright N \longrightarrow \Sigma' \triangleright N'$ and $\Sigma + n : \mathtt{T} \triangleright N \longrightarrow \Sigma' + n : \mathtt{T} \triangleright N'$. $\square$

Proposition 3.5.28 (Definability). Assume that for an arbitrary network representation Σ, the network Σ_+ denotes:

$$\Sigma_+ = \Sigma + k_0 : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\Sigma_O)], \textsc{succ} : \mathtt{ch}, \textsc{fail} : \mathtt{ch}$$

where k_0, $\textsc{succ}$ and $\textsc{fail}$ are fresh to Σ_N. Thus, for every external action μ and network representation Σ, every non-empty finite set of names Nm where $\Sigma_N \subseteq Nm$, every fresh pair of channel names $\textsc{succ}$, $\textsc{fail} \notin Nm$, and every fresh location name $k_0 \notin Nm$ connected to all observable locations in Σ_O, there exists a system $T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$ with the property that $\Sigma_+ \vdash_{\mathsf{obs}} T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$, such that:

1. $\Sigma \triangleright N \xrightarrow{\mu} \Sigma' + \mathbf{bn}(\mu) \triangleright N'$ implies
   $\Sigma_+ \triangleright N \,|\, T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma'_+ \triangleright (\nu\, \mathbf{bn}(\mu))\, N' \mid k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$
2. $\Sigma_+ \triangleright N \,|\, T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0) \Longrightarrow \Sigma'_+ \triangleright N'$,
   where $\Sigma'_+ \triangleright N' \Downarrow_{\textsc{succ}@k_0}$, $\Sigma'_+ \triangleright N' \not\Downarrow_{\textsc{fail}@k_0}$ implies that
   $N' \equiv (\nu\, \mathbf{bn}(\mu))N'' | k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$ for some N''
   such that $\Sigma \triangleright N \overset{\mu}{\Longrightarrow} \Sigma' + \mathbf{bn}(\mu) \triangleright N''$.

Proof. We have to prove that the above two clauses are true for all of the four external actions. If μ is the bound input action $(\tilde{n} : \tilde{\mathtt{L}})l : a?(V)$, where $\tilde{\mathtt{L}} = \mathsf{lnk}(\tilde{n} : \tilde{\mathtt{T}}, \Sigma)$ for some $\tilde{\mathtt{T}}$, the required system is

$$(\nu\, \tilde{n} : \tilde{\mathtt{T}})(l[\![a!\langle V\rangle.\mathsf{go}\ k_0.\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \,|\, k_0[\![\textsc{fail}!\langle\rangle]\!])$$

For the output case where μ is $(\tilde{n}:\tilde{\mathsf{L}})l : a!\langle V\rangle$, the required $T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$ is

$$k_0[\![\textsc{fail}!\langle\rangle]\!] \mid \\ l\left[\!\!\left[a?(X).(\nu\, sync)\left(\begin{array}{l} \prod_{i=1}^{m} \mathsf{if}\ x_i \notin Nm.sync!\langle\rangle \mid \prod_{j=m+1}^{|X|} \mathsf{if}\ x_j = v_j.sync!\langle\rangle \\ \mid \underbrace{sync?()..sync?()}_{|X|}.\mathsf{go}\ k_0.(\nu c)\left(\begin{array}{l} verNwStat^{\mathcal{I}}_{k_0}(x_1..x_m, c) \\ \mid c?(x).\left(\begin{array}{l}\textsc{fail}?().\textsc{succ}!\langle x_1..x_m\rangle \\ \mid \mathsf{go}\ x.\mathsf{kill}\end{array}\right)\end{array}\right)\end{array}\right)\right]\!\!\right]$$

such that

$$verNwStat^{\mathcal{I}}_{k_0}(x_1 \ldots x_m, y) \quad \Leftarrow \quad (\nu\, k' : \mathtt{T}_{k'})\mathsf{go}\ k'.(\nu d)\left(\begin{array}{l} verStat^{\mathcal{I}+(x_1..x_m:\tilde{\mathsf{K}})}_{k'}(d) \\ \mid d?().\mathsf{go}\ k_0.y!\langle k'\rangle \end{array}\right)$$

and $\mathtt{T}_{k'} = \mathtt{loc}[\mathtt{a}, Nm \cup \{x_1..x_m\}]$, $\tilde{\mathsf{K}} = \tilde{\mathsf{L}}\{^{x_1..x_m}/_{\tilde{n}}\}$

For the sake of presentation, we assume that the first $v_1 \ldots v_m$ in $V = v_1 \ldots v_{|V|}$ in μ are bound, and the remaining $v_{m+1} \ldots v_{|V|}$ are free; a more general test can be construct for arbitrary ordering of bound names in V using the same principles used for this test. We also use the conditional $\mathsf{if}\ x \notin Nm.P$ as an abbreviation for the obvious nested negative comparisons between x and each name in Nm.

The test works in two stages. Similar to the tests in [HR04, HMR04], the first stage performs the appropriate test for every input variable x_i, releasing $sync!\langle\rangle$ if the test is successful; if x_i is expected to be a bound name in μ, then we make sure it is fresh to Nm; otherwise x_i is matched with the corresponding free name. Another process waits for input on $|V|$ successful tests, that is $|V|$ inputs on the scoped channel $sync$ and then releases the code for the second stage.

The second stage deals with the verification of any new live connections and locations that become reachable as a result of the fresh names inputted. To avoid complicated routing to reach these new locations, $verNwStat^{\mathcal{I}}_{k_0}(x_1 \ldots x_m, y)$ creates a new location k' from the location k_0, with a location type that attempts to connect to any name in Nm together with the fresh bound names just inputted $x_1 \ldots x_m$; recalling Example 3.2.1, we note that the purpose of this procedure is to short-circuit our way to the newly reachable locations. We afterwards run $verStat^{\mathcal{I}+(x_1..x_m:\tilde{\mathsf{K}})}_{k'}(c)$ from this new location k', to verify that the new observable network state is indeed $\mathcal{I} + \tilde{n} : \tilde{\mathsf{L}}$. If this is the case, we signal on the continuation channel d the fresh location k', which triggers a process that goes back to location k_0 and signals once again on another continuation channel, denoted by the variable y, but eventually parameterised by the scoped channel c in the testing context above. This triggers two parallel processes; the first one consumes the barb $\textsc{fail}$ and releases an output on $\textsc{succ}$ with the bound names $x_1 \ldots x_m$, whereas the second one goes back to k' and kills it for cleaning up purposes.

In addition to bound input and bound output, we have two non-standard actions $\mathsf{kill} : l$ and $l \nleftrightarrow k$ and the test required for these actions are :

$$l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!]$$

and

$$l[\![\mathsf{break}\ k]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid (\nu\, sync)\begin{pmatrix} l[\![\mathsf{ping}\ k.\mathsf{ping}\ k.\ulcorner \mathsf{go}\ k_0.sync!\langle\rangle\urcorner]\!] \\ \mid k[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\ulcorner \mathsf{go}\ k_0.sync!\langle\rangle\urcorner]\!] \\ \mid k_0[\![sync?().sync?().\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \end{pmatrix}$$

respectively.

Since inducing faults is an asynchronous operation, the actual killing of a location or breaking of a link is independent of its observation. The observation of a kill at l is carried out from k_0 by two successive pings, first observing that l is alive and subsequently observing that l has become dead. The observation of a link break between l and k is slightly more complicated, because it needs to be tested from one of the connected locations l and k. The test is carried out, as in the previous case, through two successive pings; the first ping determines that k is accessible from l (or viceversa) while the second determine that it is not anymore. However, k (or viceversa l) can become inaccessible because it died and not because the link broke; to ensure that k (or l) became inaccessible because of a link failure, we perform the test from both endpoints, l and k, and synchronise at k_0.

We next give an outline of the proof for one of the non-standard actions, $\mathsf{kill} : l$; the proof of definability for $l \nleftrightarrow k$ is similar, whereas the proof for the remaining two actions can be extracted from [HR04, HMR04]. For the first clause, from $\Sigma \triangleright N \overset{\mathsf{kill}:l}{\longmapsto} \Sigma' \triangleright N'$ we know that $\Sigma \vdash_{\mathsf{obs}} l : \mathbf{alive}$, thus $\Sigma_+ \vdash_{\mathsf{obs}} l : \mathbf{alive}$, which means we can perform the reduction involving a positive ping:

$$\begin{aligned} \Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\ulcorner \textsc{fail}?().\textsc{succ}!\langle\rangle\urcorner]\!] \longrightarrow \\ \Sigma_+ \triangleright l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\ulcorner \textsc{fail}?().\textsc{succ}!\langle\rangle\urcorner]\!] \end{aligned} \tag{3.34}$$

From $\Sigma \triangleright N \overset{\mathsf{kill}:l}{\longrightarrow} \Sigma' \triangleright N'$, $\mathcal{I}(\Sigma_+)$ allows $\mathsf{kill} : l$ and Proposition 3.5.6 we derive

$$\Sigma_+ \triangleright N \overset{\mathsf{kill}:l}{\longrightarrow} \Sigma'_+ \triangleright N' \quad \text{where } \Sigma_+ \vdash_{\mathsf{obs}} l : \mathbf{alive} \text{ and } \Sigma'_+ \nvdash_{\mathsf{obs}} l : \mathbf{alive} \tag{3.35}$$

and from (3.35) and Lemma 3.5.25 we get

$$\Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \longrightarrow \Sigma'_+ \triangleright N'$$

and (r-par-ctxt) we derive

$$\begin{aligned} \Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\ulcorner \textsc{fail}?().\textsc{succ}!\langle\rangle\urcorner]\!] \longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\ulcorner \textsc{fail}?().\textsc{succ}!\langle\rangle\urcorner]\!] \end{aligned} \tag{3.36}$$

Subsequently we derive the sequence of reductions

$$\begin{aligned} \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\ulcorner \textsc{fail}?().\textsc{succ}!\langle\rangle\urcorner]\!] \longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow \\ \Sigma'_+ \triangleright N' \mid k_0[\![\textsc{succ}!\langle\rangle]\!] \end{aligned} \tag{3.37}$$

Combining the reductions in (3.34), (3.36) and (3.37) we prove the first clause.

For the second clause, the set of barbs $\Sigma'_+ \triangleright N' \Downarrow_{\textsc{succ}@k_0}$, $\Sigma'_+ \triangleright N' \not\Downarrow_{\textsc{fail}@k_0}$ can *only* be obtained through the sequence of reductions

$$\Sigma_+ \triangleright N \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \Longrightarrow \tag{3.38}$$

$$\Sigma^1_+ \triangleright N^1 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \longrightarrow$$

$$\Sigma^1_+ \triangleright N^1 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \Longrightarrow \tag{3.39}$$

$$\Sigma^2_+ \triangleright N^2 \mid l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \longrightarrow \tag{3.40}$$

$$\Sigma^2_+ - l \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \Longrightarrow \tag{3.41}$$

$$\Sigma^3_+ - l \triangleright N^3 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \longrightarrow$$

$$\Sigma^3_+ - l \triangleright N^3 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \Longrightarrow \tag{3.42}$$

$$\Sigma^4_+ - l \triangleright N^4 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \longrightarrow$$

$$\Sigma^4_+ - l \triangleright N^4 \mid k_0[\![\textsc{succ}!\langle\rangle]\!] \Longrightarrow \tag{3.43}$$

$$\Sigma'_+ \triangleright N' \mid k_0[\![\textsc{succ}!\langle\rangle]\!]$$

From (3.40) and Lemma 3.5.25 we deduce

$$\Sigma^2_+ \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!] \xmapsto{\mathsf{kill}:l}$$
$$\Sigma^2_+ - l \triangleright N^2 \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\ l.\lceil \textsc{fail}?().\textsc{succ}!\langle\rangle\rceil]\!]$$

and by the inductive hypothesis of (l-par-ctxt), the fact that $\mathcal{I}(\Sigma^2)$ allows $\mathsf{kill} : l$ and Proposition 3.5.6, we derive

$$\Sigma^2 \triangleright N^2 \xrightarrow{\mathsf{kill}:l} \Sigma^2 - l \triangleright N^2 \tag{3.44}$$

From (3.38), (3.39), (3.41), (3.42) and (3.43) and (r-par-ctxt) obtain

$$\begin{aligned} &\Sigma_+ \triangleright N \Longrightarrow \Sigma^1_+ \triangleright N^1 \Longrightarrow \Sigma^2_+ \triangleright N^2 \\ &\Sigma^2 - l_+ \triangleright N^2 \Longrightarrow \Sigma^3_+ \triangleright N^3 \Longrightarrow \Sigma^4_+ \triangleright N^4 \Longrightarrow \Sigma_+ \triangleright N' \end{aligned} \tag{3.45}$$

and from (3.45) and Lemma 3.5.27 we obtain

$$\begin{aligned} &\Sigma \triangleright N \Longrightarrow \Sigma^1 \triangleright N^1 \Longrightarrow \Sigma^2 \triangleright N^2 \\ &\Sigma^2 - l \triangleright N^2 \Longrightarrow \Sigma^3 \triangleright N^3 \Longrightarrow \Sigma^4 \triangleright N^4 \Longrightarrow \Sigma' \triangleright N' \end{aligned} \tag{3.46}$$

Finally, using Proposition 3.5.14 to convert the reductions in (3.46) into weak silent actions and merging these with (3.44) we obtain as required

$$\Sigma \triangleright N \overset{\mathsf{kill}:l}{\Longmapsto} \equiv \Sigma' \triangleright N' \qquad \square$$

The result of Proposition 3.5.28 means that intuitively we can provoke the action $\Sigma \triangleright N \overset{\mu}{\Longrightarrow} \Sigma' \triangleright N'$ by extending Σ with a fresh location k_0 and fresh channels $\textsc{succ}$ and $\textsc{fail}$ and placing N in parallel with $T^\mu(Nm, \textsc{succ}, \textsc{fail}, k_0)$ for a suitably chosen Nm. But in the case of actions where $\mathbf{bn}(\mu) \neq \emptyset$ we do not get precisely the residual $\Pi' \triangleright N'$ but instead $\Sigma''_+ \triangleright (\nu\, \mathbf{bn}(\mu))\, N \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$where $\Sigma'' + \mathbf{bn}(\mu) = \Sigma'$. We therefore state and prove a variant the extrusion lemma in [HR04, HMR04], which enables us to recover the residual $\Sigma' \triangleright N'$ from $\Sigma''_+ \triangleright (\nu\, \mathbf{bn}(\mu))N \,|\, k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu)\rangle]\!]$; this lemma uses the preliminary lemma below, which we chose to extract as an important step of the proof.

Lemma 3.5.29. Suppose δ, k_0 are fresh to the systems $M, k[\![P(X)]\!]$. Suppose also that $k \in \mathtt{C}$. Then:

$$\Sigma \models (\nu\,\tilde{n}:\tilde{\mathtt{T}})(M \,|\, k[\![P(\tilde{n})]\!]) \;\cong$$
$$(\nu\,\tilde{n}:\tilde{\mathtt{T}})(\nu\,\delta:\mathtt{ch})(\nu\,k_0:\mathtt{loc[a,C]})(M \,|\, k_0[\![\delta!\langle\tilde{n}\rangle]\!] \,|\, k_0[\![\delta?(X).\mathsf{go}\;k.P(X)]\!])$$

Proof. We note that the left hand system can be obtained from the right hand system in two reductions, communication on δ and migrating from k_0 to k, that cannot be interfered with by any context. It is easy to come up with a bisimulation proving that the two systems are reduction barbed congruent. □

Lemma 3.5.30 (Extrusion). Suppose SUCC, FAIL, k_0 are fresh to the network representations Σ^M, Σ^N, M and N. Then

$$\mathcal{I} \models \Sigma^M_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \cong \Sigma^N_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!]$$
$$\text{implies } \Sigma^M + \tilde{n}:\tilde{\mathtt{T}} \triangleright M \cong \Sigma^N + \tilde{n}:\tilde{\mathtt{U}} \triangleright N$$

Proof. We define the relation $\mathcal{R}$ as:

$$\mathcal{R} \;=\; \left\{ \langle \Sigma^M + \tilde{n}:\tilde{\mathtt{T}} \triangleright M, \Sigma^N + \tilde{n}:\tilde{\mathtt{U}} \triangleright N\rangle \;\middle|\; \begin{array}{l} \Sigma^M_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \cong \\ \qquad \Sigma^N_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \end{array} \right\}$$

and show that $\mathcal{R}$ satisfies the defining properties of $\cong$. It is obviously reduction closed. We here outline the proof for the barb preserving and contextuality properties.

Suppose $\Sigma^M + \tilde{n} : \tilde{\mathtt{T}} \triangleright M \;\;\mathcal{R}\;\; \Sigma^N + \tilde{n} : \tilde{\mathtt{U}} \triangleright N$ and $\Sigma^M + \tilde{n} : \tilde{\mathtt{T}} \triangleright M \Downarrow_{a@l}$; we have to show $\Sigma^N + \tilde{n} : \tilde{\mathtt{U}} \triangleright N \Downarrow_{a@l}$. If $l, a \notin \tilde{n}$ this is straightforward since in this case $\Sigma^M_+ \triangleright (\nu\,\tilde{n} : \tilde{\mathtt{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \Downarrow_{a@l}$, by barb preserving, $\Sigma^N_+ \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \Downarrow_{a@l}$ which can only be because $\Sigma^N + \tilde{n}:\tilde{\mathtt{U}} \triangleright N \Downarrow_{a@l}$. So suppose, as an example, that $a \in \tilde{n}$. Even though we no longer have that $\Sigma^M_+ \triangleright (\nu\,\tilde{n} : \tilde{\mathtt{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \Downarrow_{a@l}$, the restricted name a can be extruded via SUCC through the system:

$$T_a \;\;\Leftarrow\;\; k_0[\![\textsc{succ}?(X).\mathsf{mv}\;l(X_a?().\mathsf{go}\;k_0.\delta!\langle\rangle)]\!]$$

where δ is a fresh channel and X_a is the variable x_i where a is bound on input. Since $\Sigma^M \triangleright M \Downarrow_{a@l}$ it follows that

$$\Sigma^M_+ + \delta:\mathtt{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a \Downarrow_{\delta@k_0}$$

From the definition of $\cong$, we know

$$\Sigma^M_+ + \delta:\mathtt{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a \cong \Sigma^N_+ + \delta:\mathtt{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a$$

and by barb preservation we conclude

$$\Sigma^N_+ + \delta:\mathtt{ch} \triangleright (\nu\,\tilde{n}:\tilde{\mathtt{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \,|\, T_a \Downarrow_{\delta@k_0}$$

which only be because $\Sigma^N \triangleright N \Downarrow_{a@l}$ as required.

The case for when $n = l$ is similar, only that instead of T_a we use the system:

$$T_l \;\;\Leftarrow\;\; k_0[\![\textsc{succ}?(X).(\nu\,k:(\mathbf{dom}(\Sigma'_O)\cup X_l))\mathsf{go}\;k, X_l.a?().\mathsf{go}\;k, k_0.\delta!\langle\rangle]\!]$$

This system is similar to T_a with the exception that a specific location k is created so that we short-circuit our route to l, similar to the procedure we used earlier in the definability proof of bound outputs (see Proposition 3.5.28).

We still have to show that $\mathcal{R}$ is contextual. As an example we show that it is preserved by parallel system contexts and leave the simpler case, that for network extensions, to the interested reader. Suppose $\mathcal{I} \models \Sigma^M \triangleright M \;\mathcal{R}\; \Sigma^N \triangleright N$; we have to show that for arbitrary $k[\![P]\!]$ such that $\mathcal{I} \vdash k[\![P]\!]$ then we have $\mathcal{I} \models \Sigma^M \triangleright M \,|\, k[\![P]\!] \;\mathcal{R}\; \Sigma^N \triangleright N \,|\, k[\![P]\!]$.

By definition of $\mathcal{R}$, we have $\mathcal{I} \models \Sigma^M \triangleright M \;\mathcal{R}\; \Sigma^N \triangleright N$ because

$$\mathcal{I}' \models \Sigma^M_+ \triangleright (\nu\, \tilde{n}\!:\!\tilde{\mathrm{T}})M|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \;\cong\; \Sigma^N_+ \triangleright (\nu\, \tilde{n}\!:\!\tilde{\mathrm{U}})N|k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \tag{3.47}$$

We define the system

$$T_{k[\![P]\!]} \;\Leftarrow\; k_0[\![\textsc{succ}?(X).\mathsf{go}\; k_0'.\delta!\langle X\rangle \,|\, (X)\mathsf{go}\; k.P]\!]$$

where δ, k_0' are fresh names and $(X)\mathsf{go}\; k.P$ substitutes all occurrences of $\tilde{n}$ in $\mathsf{go}\; k.P$ by the appropriate variables $x_i \in X$. From $\mathcal{I} \vdash k[\![P]\!]$ we deduce that $\mathcal{I}'' \vdash T_{k[\![P]\!]}$ for $\mathcal{I}'' = \mathcal{I}' + \delta : \mathtt{ch} + k_0' : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\mathcal{I}'_O)]$ and subsequently, by contextuality of $\cong$ and (3.47), we obtain

$$\mathcal{I}'' \models \Sigma^M_{++} \triangleright M' \,|\, T_{k[\![P]\!]} \;\cong\; \Sigma^N_{++} \triangleright N' \,|\, T_{k[\![P]\!]} \tag{3.48}$$

where

$$\begin{aligned} M' &= (\nu\, \tilde{n}\!:\!\tilde{\mathrm{T}})M \,|\, k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \\ N' &= (\nu\, \tilde{n}\!:\!\tilde{\mathrm{U}})N \,|\, k_0[\![\textsc{succ}!\langle\tilde{n}\rangle]\!] \\ \Sigma^M_{++} &= \Sigma^M_+ + \delta : \mathtt{ch} + k_0' : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\mathcal{I}'_O)] \\ \Sigma^N_{++} &= \Sigma^N_+ + \delta : \mathtt{ch} + k_0' : \mathtt{loc}[\mathtt{a}, \mathbf{dom}(\mathcal{I}'_O)] \end{aligned}$$

From (3.48) and Proposition 3.5.24 we deduce that we can scope $\textsc{succ}$ and k_0 to obtain

$$\mathcal{I}' \models \Sigma^M_+ \triangleright (\nu\, \textsc{succ}, k_0)M' \,|\, T_{k[\![P]\!]} \;\cong\; \Sigma^N_+ \triangleright (\nu\, \textsc{succ}, k_0)N' \,|\, T_{k[\![P]\!]} \tag{3.49}$$

and by Lemma 3.5.29 and we get

$$\mathcal{I}' \models \Sigma^M_+ \triangleright (\nu\, \tilde{n}\!:\!\tilde{\mathrm{T}})M \,|\, k[\![P]\!] \,|\, k_0'[\![\delta!\langle\tilde{n}\rangle]\!] \;\cong\; \Sigma^N_+ \triangleright (\nu\, \tilde{n}\!:\!\tilde{\mathrm{T}})N \,|\, k[\![P]\!] \,|\, k_0'[\![\delta!\langle\tilde{n}\rangle]\!] \tag{3.50}$$

from which, by definition of $\mathcal{R}$, we derive $\mathcal{I} \models \Sigma^M \triangleright M|k[\![P]\!] \;\mathcal{R}\; \Sigma^N \triangleright N|k[\![P]\!]$ as required. □

Proposition 3.5.31 (Completness).

$$\mathcal{I} \models \Sigma^1 \triangleright M_1 \cong \Sigma^2 \triangleright M_2 \;\text{ implies }\; \mathcal{I} \models \Sigma^1 \triangleright M_1 \approx \Sigma^2 \triangleright M_2$$

Proof. Suppose $\Sigma^1 \triangleright M_1 \overset{\mu}{\longmapsto} \Sigma^1_1 \triangleright M_1'$; we must find a move $\Sigma^2 \triangleright M_2 \overset{\widehat{\mu}}{\Longmapsto} \Sigma^2_1 \triangleright M_2'$ such that $\Sigma^1_1 \triangleright M_1' \;\cong\; \Sigma^2_1 \triangleright M_2'$. If μ is an internal move then the matching move is obtained from the fact that $\cong$ is reduction closed, together with Proposition 3.5.14. If μ is an external action, then by choosing Nm so that it contains all the free names in $\mathcal{I}_N$ and

choosing fresh SUCC, FAIL, k_0, from the first part of Proposition 3.5.28 and the assumption $\Sigma^1 \triangleright M_1 \overset{\mu}{\longmapsto} \Sigma^1_1 + \mathbf{bn}(\mu) \triangleright M'_1$ we obtain

$$\Sigma^1_+ \triangleright M_1 | T^\mu(Nm, \text{SUCC}, \text{FAIL}, k_0) \Longrightarrow \Sigma^1_{1+} \triangleright (\nu\, \mathbf{bn}(\mu)) M'_1 \,|\, k_0[\![\text{SUCC}!\langle \mathbf{bn}(\mu)\rangle]\!]$$

By contextuality and reduction closure of $\cong$, we know that there is a matching move

$$\Sigma^2_+ \triangleright M_2 | T^\mu(Nm, \text{SUCC}, \text{FAIL}, k_0) \Longrightarrow \Sigma \triangleright N$$

for some $\Sigma \triangleright N$ such that $\Sigma^1_{1+} \triangleright (\nu\, \mathbf{bn}(\mu)) M'_1 \,|\, k_0[\![\text{SUCC}!\langle \mathbf{bn}(\mu)\rangle]\!] \cong \Sigma \triangleright N$. This in turn means that $\Sigma \triangleright N \Downarrow_{\text{SUCC}@k_0}$ and $\Sigma \triangleright N \not\Downarrow_{\text{FAIL}@k_0}$ and so the second part of Proposition 3.5.28 now gives that $\Sigma \triangleright N \equiv \Sigma^2_{1+} \triangleright (\nu\, \mathbf{bn}(\mu)) M'_2 \,|\, k_0[\![\text{SUCC}!\langle \mathbf{bn}(\mu)\rangle]\!]$ for some Σ^2_{1+}, M'_2 such that $\Sigma^2 \triangleright M_2 \overset{\mu}{\Longmapsto} \Sigma^2_1 + \mathbf{bn}(\mu) \triangleright M'_2$. This is the required matching move, since the Extrusion Lemma 3.5.30, gives us the required

$$\Sigma^1_1 + \mathbf{bn}(\mu) \triangleright M'_1 \cong \Sigma^2_1 + \mathbf{bn}(\mu) \triangleright M'_2 \qquad \Box$$

3.6 Summary

In this chapter we have carried over the work presented in Chapter 2 and extended it to obtain a simple adaptation of Dπ, in which there is an explicit representation of the underlying network on which processes execute, exhibiting both *node and link failures*; a core aspect of this adaptation is the encoding of node status and connections as type information. We then defined a reduction semantics to describe the behaviour of systems in the presence of node and link failures; afterwards we applied techniques for actions dependent on the observer's knowledge, developed for the π-calculus in [HR04] and Dπ in [HMR04], to characterise a natural notion of barbed congruence. Our main result is a *fully-abstract* bisimulation equivalence with which we can reason about the behaviour of systems in the presence of dynamic network failures. To the best of our knowledge, this is the first time system behaviour in the presence of *link* failure has ever been investigated.

We chose to develop the theory in terms of the calculus itself, despite the widely held view that representation of nodes *only* is sufficient; this would typically entail encoding a link between location l and k as an intermediary node lk, encoding migration from l to k as a two step migration from l to lk and lk to k and finally encoding link failure as the intermediary node lk failing. We believe that a calculus with partial connection between nodes is very natural in itself since WANs are often *not* a clique. Our calculus also gives rise to an interesting theory of partial views that we believe deserves to be investigated in its own right. In addition, we also wanted to explore the interplay between node and link failure and their respective observation from the software's point of view. With these points in mind, we postulate that any such encoding of DπF- expressing both node and link failure - in terms of a node failure only calculus would be cumbersome to use and the corresponding theory of partial views would be too complicated to develop. Moreover, it is unlikely that this resultant theory would be fully abstract, due to the fact that any encoding would typically decomposes atomic reductions such as migration into sub-reductions, which in turn affects the resulting bisimulation equivalence; see [GG89].

Rather than being a body of work that could be directly applied to real case scenarios, we believe that the work in this Chapter is best viewed as a succinct well-founded framework from which numerous variations could be considered. Having said this, we feel that our framework, as it currently stands, lends itself well to the study of distributed software that needs to be aware of the *dynamic* computing context in which it is executing; various examples can be drawn from ad-hoc networks, embedded systems and generic routing software. In these settings, the software typically *discovers* new parts of the neighbouring network at runtime and *updates* its knowledge of the current underlying network with changes caused by failure.

+

Chapter 4

Fault Tolerance

In this chapter we carry over the work done in Chapter 2 to study fault tolerant systems in DπLoc, a distributed calculus that expressed node failure only. In § 4.2, we introduce a typed version of the language called *typed DπLoc* , where the types are used to define a restricted observer view of the system. This typed restricted observer view is then used as a foundation for the definition of fault tolerance, which we develop in § 4.3. We subsequently develop bisimulation techniques in § 4.4 that enable us to prove fault tolerant properties of systems.

4.1 Motivation

Distributed computing lends itself very well to fault tolerance because distribution yields natural notions of *partial failure*, that is faults that affect a *subset* of the computation. *Faults* inherent to distribution, such as unstable connections between locations and dead locations without processing power, lead to *failure* such as unreliable migration across locations and halted computation limited to the faulty location, respectively. As a result, this setting also provides scope for space and time redundancy across multiple locations in terms of replication.

With reference to the review of § 1.4, in the sequel we base our discussion on simple examples using *stateless* replicas that are invoked only once; these can alternatively be viewed as read-only replicas. This simplification obviates the need for additional machinery to sequence multiple requests (in the case of active replication) or synchronise the state of replicas (in the case of passive replication); as a result management techniques based on lazy replication simply collapse into passive replication category. Nevertheless, these simple examples still capture the essence of the concepts we chose to study.

Example 4.1.1. Consider the DπLoc systems $\mathsf{server}_1..\mathsf{server}_3$, three server implementations similar to the ones seen earlier in Example 2.2.9, accepting client requests on channel *req* with two arguments, x being the value to process and y being the channel name on

which the answer is returned:

$$\mathsf{server}_1 \;\Leftarrow\; (\nu\, data)\left(\begin{array}{l} l[\![req?(x,y).\mathsf{go}\; k_1.data!\langle x,y,l\rangle]\!] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\; z.y!\langle f(x)\rangle]\!] \end{array}\right)$$

$$\mathsf{server}_2 \;\Leftarrow\; (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu sync)\left(\begin{array}{l} \mathsf{go}\; k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\; k_2.data!\langle x, sync, l\rangle \\ \mid sync?(x).y!\langle x\rangle \end{array}\right)\right]\!\!\right] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\; z.y!\langle f(x)\rangle]\!] \\ \mid k_2[\![data?(x,y,z).\mathsf{go}\; z.y!\langle f(x)\rangle]\!] \end{array}\right)$$

$$\mathsf{server}_3 \;\Leftarrow\; (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[req?(x,y).(\nu sync)\left(\begin{array}{l} \mathsf{go}\; k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\; k_2.data!\langle x, sync, l\rangle \\ \mid \mathsf{go}\; k_3.data!\langle x, sync, l\rangle \\ \mid sync?(x).y!\langle x\rangle \end{array}\right)\right]\!\!\right] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\; z.y!\langle f(x)\rangle]\!] \\ \mid k_2[\![data?(x,y,z).\mathsf{go}\; z.y!\langle f(x)\rangle]\!] \\ \mid k_3[\![data?(x,y,z).\mathsf{go}\; z.y!\langle f(x)\rangle]\!] \end{array}\right)$$

The theory developed in Chapter 2 enabled us to differentiate between these three implementations, based on the different behaviour observed when put in parallel with systems such as

$$\mathsf{client} \;\Leftarrow\; l[\![req!\langle n, ret\rangle]\!]$$

In this Chapter, we would like to say more about these three implementations. In particular, whereas server_1 holds a *single* remote copy of its database, server_2 and server_3 keep multiple *redundant* copies distributed at k_1, k_2 and k_3; moreover, they employ them is specific manner to reduce the *dependency* on these remote locations. Stated otherwise, in a setting where locations may incur faults, server_2 seems to be more *fault tolerant* than server_1 because it *preserves* its *behaviour* when at most *one* (location) fault is induced in a network with no faults. More formally, if we consider the network representation used in § 2 of the form $\Pi = \langle \mathcal{N}, \mathcal{A}\rangle$, where $\mathcal{N} = \mathcal{A} = \{l, k_1, k_2, k_3\}$, that is all the locations are alive, we are always guaranteed that

$$\text{for } i = 1..3,\ \forall\, \Pi', N \text{ if } \quad \Pi \triangleright \mathsf{server}_2 \,|\, \mathsf{client} \,|\, k_i[\![\mathsf{kill}]\!] \longrightarrow^* \Pi' \triangleright N \quad \text{then } \ \Pi' \triangleright N \Downarrow_{ret@l}$$

In contrast,

$$\exists\, \Pi', N \text{ such that } \quad \Pi \triangleright \mathsf{server}_1 \,|\, \mathsf{client} \,|\, k_1[\![\mathsf{kill}]\!] \longrightarrow^* \Pi' \triangleright N \ \text{ where } \ \Pi' \triangleright N \not\Downarrow_{ret@l}$$

Similarly, server_3 is more *fault tolerant* than server_1 and server_2 because it preserves its behaviour up to *two* faults induced, that is

$$\begin{array}{l} \text{for } i, j = 1..3, \text{ we have} \\ \quad \forall\, \Pi', N \text{ if } \quad \Pi \triangleright \mathsf{server}_3 \,|\, \mathsf{client} \,|\, k_i[\![\mathsf{kill}]\!] \,|\, k_j[\![\mathsf{kill}]\!] \longrightarrow^* \Pi' \triangleright N \quad \text{then } \ \Pi' \triangleright N \Downarrow_{ret@l} \end{array}$$

whereas

for any $j = 1..3 \quad \exists\, \Pi', N$ such that

$$\Pi \triangleright \mathsf{server}_1 \,|\, \mathsf{client} \,|\, k_1[\![\mathsf{kill}]\!] \,|\, k_j[\![\mathsf{kill}]\!] \quad \longrightarrow^* \quad \Pi' \triangleright N \text{ where } \Pi' \triangleright N \not\Downarrow_{ret@l}$$

$\exists\, \Pi', N$ such that

$$\Pi \triangleright \mathsf{server}_2 \,|\, \mathsf{client} \,|\, k_1[\![\mathsf{kill}]\!] \,|\, k_2[\![\mathsf{kill}]\!] \quad \longrightarrow^* \quad \Pi' \triangleright N \text{ where } \Pi' \triangleright N \not\Downarrow_{ret@l}$$

One way to envisage fault tolerance for the above configurations would be to use a mechanism that separates a configuration into two parts: the *observable* part and *hidden* part. Intuitively, fault tolerance would be defined with respect to observations such as barbs, limited to the observable part; the hidden part of the configuration would delineate where redundancy is held and where faults occur.

In the sequel, we define the observable and hidden parts of a configuration by partitioning the potentially failing entities, in our case locations, into two groups: *public* and *confined*. The observable part of a configuration is defined in terms of the processes located at *public* locations; a valid observer is therefore allowed to place code at public locations *only*. The hidden part of a configuration is similarly defined in terms of processes located at *confined* locations; thus, redundancy would be located at confined locations. Subsequently, our fault tolerance definition would be based on observations made on the public part and would assume that faults are only induced on the confined part. Hence, in Example 4.1.1, location l would be set as public whereas $k_{1..3}$ would be set as confined. The intuitions behind this setup are twofold:

- the public locations are *dependable* locations: we can assume, up to a high degree of certainty, that they will not fail. Based on this dependability assumption, we allow the observer to interact with the system through these locations.
- the confined locations are assume to be *undependable*, or else locations where we do not have any guarantee that they will not fail. It seems natural to prohibit the observer from using these locations, or else, only assume responsibility for observers that use public locations. In accordance, we assume that these are the only locations that can fail and these are the locations where we place redundancy to be able to tolerate these faults.

It also seems natural to limit our definition of fault tolerance to configurations that observe this delineation. The rationale would be that valid obeserver that limit interactions to reliable resources never end up using unrealiable resources as a result of interaction. Stated otherwise, valid observers only expect to learn fresh *reliable* resource names through scope extrusion. We explain this further in the following example.

Example 4.1.2. [Breaking the delineation] We have so far argued, at least informally, that server_2 in Example 4.1.1, exhibits fault tolerant properties because it preserves certain observable behaviour when faults are induced. For instance, for observers such as client, limited to the public location l, we have the property

$$\forall\, \Pi', N \text{ if } \quad \Pi \triangleright \mathsf{server}_2 \,|\, \mathsf{client} \longrightarrow^* \Pi' \triangleright N \quad \text{then} \quad \Pi' \triangleright N \Downarrow_{ret@l}$$

and this property is preserved if one fault is induced,

$$\text{for } i = 1..3,\ \forall\, \Pi', N \text{ if } \quad \Pi \triangleright \mathsf{server}_2 \,|\, \mathsf{client} \,|\, k_i[\![\mathsf{kill}]\!] \longrightarrow^* \Pi' \triangleright N \quad \text{then } \ \Pi' \triangleright N \Downarrow_{ret@l}$$

In a value passing calculus such as DπLoc, we may however have the variant server:

$$\mathsf{server}_2' \Leftarrow \mathsf{server}_2 \,|\, l[\![a!\langle k_1 \rangle]\!]$$

which breaks the delineation between the public locations (used by the system and the observer) and confined locations (used only by the system). We note that server_2' communicates the confined value k_1 at a public location l and this value may be obtained by a valid observer listening on channel a at the public location l. More specifically, if we consider the valid observer located at l

$$\mathsf{client}' \Leftarrow \mathsf{client} \,|\, l[\![a?(x).Q(x)]\!]$$

where $Q(x)$ consists of public names only, then we have the reduction

$$\Pi \triangleright \mathsf{server}_2' \,|\, \mathsf{client}' \longrightarrow^* \Pi' \triangleright \mathsf{server}_2 \,|\, \mathsf{client} \,|\, l[\![Q(k_1)]\!]$$

where the resulting observer, $\mathsf{client} \,|\, l[\![Q(k_1)]\!]$, is now also defined in terms of the confined location k_1. This may interfere with our intuition of fault tolerance so far. If for example $Q(x) = \mathsf{ping}\ x.b!\langle\rangle\lceil c!\langle\rangle\rceil$, the process $Q(k)$ may perform simple observations from l that change when failure occurs. Using the previous argument, in a setting with no faults, we have the behaviour

$$\forall\, \Pi', N \text{ if } \quad \Pi \triangleright \mathsf{server}_2' \,|\, \mathsf{client}' \longrightarrow^* \Pi' \triangleright N \quad \text{then } \ \Pi' \triangleright N \Downarrow_{b@l} \ \text{ and} \Pi' \triangleright N \not\Downarrow_{c@l}$$

On the other hand, when we induce a fault at k_1 we may have the sequence of reductions where

$$\exists\, \Pi', N \text{ such that } \quad \Pi \triangleright \mathsf{server}_2' \,|\, \mathsf{client}' \,|\, k_i[\![\mathsf{kill}]\!] \longrightarrow^* \Pi' \triangleright N \quad \text{where } \ \Pi' \triangleright N \Downarrow_{c@l} \ \text{ and} \Pi' \triangleright N \not\Downarrow_{b@l}$$

Even worse, if $Q(x) \stackrel{\text{def}}{=} \mathsf{go}\ x.\mathsf{kill}$, then the resultant $\mathsf{client} \,|\, l[\![Q(k_1)]\!]$ would induce an additional fault which would interfere with the observation obtained from the server. More specifically, while

$$\forall\, \Pi', N \text{ if } \quad \Pi \triangleright \mathsf{server}_2' \,|\, \mathsf{client}' \longrightarrow^* \Pi' \triangleright N \quad \text{then } \ \Pi' \triangleright N \Downarrow_{ret@l}$$

for the case were we try to induce a single fault at k_2, we now have

$$\exists\, \Pi', N \text{ such that } \quad \Pi \triangleright \mathsf{server}_2' \,|\, \mathsf{client}' \,|\, k_2[\![\mathsf{kill}]\!] \longrightarrow^* \Pi' \triangleright N \quad \text{where } \ \Pi' \triangleright N \not\Downarrow_{ret@l}$$

since the client induced a second fault at k_1. ■

In the sequel, we limit our definition of fault tolerance to configurations thatnever communicate confined values to observers. More specifically, in the next section, we enrich DπLoc with a type system that uses types to partition resources into public and confined and guarantees that all well formed configurations never violate the delineation between the public part and the confined parts.

Table 15. *Syntax of typed DπLoc*

Types

$\mathtt{T,U}$	::=	$\mathtt{loc[B,S]} \mid \mathtt{ch}\langle\mathtt{B},\tilde{\mathtt{W}}\rangle$	$\mathtt{S}$	::=	$\mathtt{a} \mid \mathtt{d}$
$\mathtt{W,Z}$	::=	$\mathtt{loc[B]} \mid \mathtt{ch}\langle\mathtt{B},\tilde{\mathtt{W}}\rangle$	$\mathtt{B}$	::=	$\mathtt{p} \mid \mathtt{c}$

Processes

$$P,Q \quad ::= \quad \ldots \quad \mid (\nu\, n\!:\!\mathtt{T})P \quad \mid \ldots$$

Systems

$$M,N,O \quad ::= \quad \ldots \quad \mid (\nu\, n\!:\!\mathtt{T})N$$

4.2 Typed DπLoc

In Chapter 2 we used types to separate between channel names and location names and also represent the state of locations: we used `ch` to denote channel names and `loc[A]` to denote locations with state `A`, ranging over `a` (alive) or `d` (dead). Typed DπLoc is an extension of DπLoc, where we enrich the type system to attain two aims:

1. We create a *restricted* observer view of the configuration based on typing (as opposed to a restricted view based on reachability as in Chapter 3.
2. We guarantee that this view is never violated during any sequence of reductions with observers that respect this restricted view.

We introduce the notion of *stateless boundary* types, ranged over by `W,Z` in Table 15, which state whether a channel or a location name is *public* and *confined*, using the respective tags `p` and `c`. Using this classification, we can create our restricted view of the configuration: as the name suggests, public names can be used publicly, outside the scope of a configuration by observers, whereas the use of confined names is limited to within the configuration. Boundaries are not only set on location types, `loc[B]`, as motivated earlier in § 4.1, but also on channel types; moreover, the object type of channels, $\tilde{\mathtt{W}}$, are also specified and thus a channel is denoted as $\mathtt{ch}\langle\mathtt{B},\tilde{\mathtt{W}}\rangle$. These additions are required to allow the communication of confined values in a controlled manner and thus enabling us to ensure that the observer restricted view is preserved.

Stateless boundary types are merged with the previous simple state types used in Chapter 2 to obtain *stateful boundary* types, ranged over by `T,U` in Table 15. The only difference between stateless and stateful boundary types are the way location types are denoted: stateful location boundary types are represented as `loc[B,A]` where we also represent the state of the location `A`; by contrast stateless location boundary types are simply denoted as `loc[B]`. The channel types are the same for both stateful and stateless boundary types. In Table 15, processes, P,Q, and systems, M,N, use *stateful* types, declaring the state of the resource and its boundary level.

Definition 4.2.1 (Well Formed Typed Configuration). Systems in typed DπLoc are subject to a *typed* network representation, Γ, a tuple of the form $\langle \mathcal{T}, \mathcal{A} \rangle$ where

- $\mathcal{T}$ is a set of tuples, $\{n_1 : \mathtt{W}_1, \ldots, n_m : \mathtt{W}_m\}$, associating a stateless type to every defined name in the network
- $\mathcal{A}$ is the liveset as before, consisting of a set of locations that are alive in the network, $\mathcal{A} \subseteq \mathbf{loc}(\mathcal{T})$.

The type system of typed DπLoc restricts well formed types through the use of purely public types.

- A *purely public stateless* type, denoted by $\mathtt{P}$, is defined as

$$\mathtt{P} \;=\; \mathtt{loc[p]} \mid \mathtt{ch}\langle \mathtt{p}, \tilde{\mathtt{P}} \rangle$$

  and excludes public channels that communicate confined values.
- A *well-formed* type in typed DπLoc is either a (stateful or stateless) location type, $\mathtt{loc[B,S]}$ or $\mathtt{loc[B]}$, a confined channel, $\mathtt{ch}\langle \mathtt{c}, \tilde{\mathtt{W}} \rangle$ or a purely public channel type $\mathtt{ch}\langle \mathtt{p}, \tilde{\mathtt{P}} \rangle$.

A well-formed typed network representation Γ, is one such that every type used in $\Gamma_{\mathcal{T}}$ is a well-formed type. We say a configuration $\Gamma \triangleright N$ in typed DπLoc is well-formed if:

- Γ is a well-formed typed network representation.
- N used only well-formed *stateful* types and is valid with respect to Γ, denoted as $\Gamma \vdash N$, and defined in terms of the typing rules in Table 16.

Finally, an observer system O is valid with respect to a typed network representation Γ, denoted as $\Gamma \vdash_{\mathsf{obs}} O$, if it is well typed with respect to the public names declared in Γ, denoted as $\mathbf{pub}(\Gamma)$ in Table 16; see Appendix for definition. ■

The restriction on a well-formed configuration, imposed through purely public types, is used to ensure that the public and confined views are never violated during reduction (Example 4.1.2). Intuitively, since configurations can only use purely public types, then by the definition of purely public channels, their object type is always a list of public types: if the typing discipline is observed and the types of the values communicated on a channel is matched with the object type, then we are guaranteed that no confined value is ever communicated on a public channel. As a consequence, since observers are only allowed to use public channels, they can never obtain any confined valued from purely public types. In fact, a public channel used for communicating confined values does not make much sense in our calculus because:

- an observer, limited only to public values, can never output on such a channel because it does not have confined values to match the object type.
- an observer should not be allowed to input on such a channel because it would obtain confined values, thereby violating the public view of the configuration.

As a result, whenever the system needs to communicate confined values, it needs to do so using confined channel; as we have just mentioned, this however does not affect the observer in any way.

The typing rules in Table 16, merely ensure that the typing in Γ is observed by the system N. Thus, (t-out) checks that the output value matches the object type of the channel whereas (t-in-rep) and (t-rest) ensure that the process P uses its bound variables according to the type assigned to them; the remaining rules are straightforward. In fact, we note that the only rules where boundary tags are used are in rules (t-out) and (t-in-rep). In the typing rules of Table 16 we make extensive use of the partial operator for extending networks, defined for fresh names n in Γ as:

$$\Gamma + n:\mathtt{T} \stackrel{\text{def}}{=} \begin{cases} \langle \mathcal{T} \cup \{n:\mathtt{ch}\langle \mathtt{B}, \tilde{\mathtt{W}}\rangle\}, \mathcal{A}\rangle & \text{if } \Gamma = \langle \mathcal{T}, \mathcal{A}\rangle \text{ and } \mathtt{T} = \mathtt{ch}\langle \mathtt{B}, \tilde{\mathtt{W}}\rangle \\ \langle \mathcal{T} \cup \{n:\mathtt{loc}[\mathtt{B}]\}, \mathcal{A}\rangle & \text{if } \Gamma = \langle \mathcal{T}, \mathcal{A}\rangle \text{ and } \mathtt{T} = \mathtt{loc}[\mathtt{B}, \mathtt{d}] \\ \langle \mathcal{T} \cup \{n:\mathtt{loc}[\mathtt{B}]\}, \mathcal{A} \cup \{n\}\rangle & \text{if } \Gamma = \langle \mathcal{T}, \mathcal{A}\rangle \text{ and } \mathtt{T} = \mathtt{loc}[\mathtt{B}, \mathtt{a}] \end{cases}$$

We also make extensive use of the type judgement $\Gamma \vdash n:\mathtt{T}$ defined as before as

$$\Gamma + n:\mathtt{T} \vdash n:\mathtt{T}$$

This also means that values have a unique type according to a typed network representation Γ. This property is formalised in the follow lemma and assume in every proof in the sequel using type judgements.

Lemma 4.2.2 (Type Uniqueness). If $\Gamma \vdash n:\mathtt{T}$ and $\Gamma \vdash n:\mathtt{U}$ then $\mathtt{T} = \mathtt{U}$.

Proof. Immediate from the definitions of $\Gamma \vdash n:\mathtt{T}$ and $\Gamma + n:\mathtt{T}$ □

Notation 4.2.3. In Table 16 and the rest of the Chapter, we use a number of abbreviations for types in typed DπLoc. We use $\mathtt{B}[\mathtt{S}]$ and $\mathtt{B}\langle\tilde{\mathtt{W}}\rangle$ as a shorthand for the location type $\mathtt{loc}[\mathtt{B},\mathtt{S}]$ and channel type $\mathtt{ch}\langle\mathtt{B},\tilde{\mathtt{W}}\rangle$ respectively. We omit the boundary level from the types and use $\mathtt{loc}[\mathtt{S}]$, $\mathtt{loc}$ and $\mathtt{ch}\langle\tilde{\mathtt{W}}\rangle$ when the boundary level is not important, we omit the location state type, $\mathtt{B}[-]$, and the channel object type, $\mathtt{B}\langle-\rangle$, when these are irrelevant and use $\mathtt{loc}$, $\mathtt{ch}$ when we simply want to distinguish between a location and a channel name. In addition, when we are only concerned about the boundary level of a type, we use $\mathtt{p}$, $\mathtt{c}$ as a shorthand for $\mathtt{p}[\mathtt{S}]$, $\mathtt{p}\langle\tilde{\mathtt{T}}\rangle$ and $\mathtt{c}[\mathtt{S}]$, $\mathtt{c}\langle\tilde{\mathtt{T}}\rangle$ respectively. Finally, we also use extensively the judgment $\Gamma \vdash_{\mathsf{obs}} n$ to denote $\Gamma \vdash n : \mathtt{p}$ and $\Gamma \vdash_{\mathsf{obs}} \mathtt{T}$ to denote that $\mathtt{T}$ is purely public, that is $\mathtt{T} = \mathtt{p}[-]$ or $\mathtt{T} = \mathtt{p}\langle\tilde{\mathtt{P}}\rangle$; in doing so, we standardise our notation with that used in the preceding two chapters. ■

4.2.1 Reduction Semantics

The reductions for typed DπLoc configurations take the form

$$\Gamma \triangleright N \longrightarrow \Gamma' \triangleright N'$$

and are defined in terms of the reduction rules in Tables 2, 4 and 3 given earlier in Chapter 2, where we simply substitute the former network representation, Π, with the

Table 16. *Typing rules for typed DπLoc*

Systems

(t-rest)
$$\frac{\Gamma + \tilde{n} : \tilde{\mathtt{T}} \vdash N}{\Gamma \vdash (\nu\, \tilde{n} : \tilde{\mathtt{T}})N}$$

(t-par)
$$\frac{\Gamma \vdash N,\ M}{\Gamma \vdash N|M}$$

(t-proc)
$$\frac{\begin{array}{l}\Gamma \vdash l : \mathtt{loc}\\ \Gamma \vdash P\end{array}}{\Gamma \vdash l[\![P]\!]}$$

Processes

(t-out)
$$\frac{\begin{array}{l}\Gamma \vdash u : \mathsf{ch}\langle \mathtt{T}\rangle\\ \Gamma \vdash V : \mathtt{T}\\ \Gamma \vdash P\end{array}}{\Gamma \vdash u!\langle V\rangle.P}$$

(t-in-rep)
$$\frac{\begin{array}{l}\Gamma \vdash u : \mathsf{ch}\langle \mathtt{T}\rangle\\ \Gamma + X{:}\mathtt{T} \vdash P\end{array}}{\begin{array}{l}\Gamma \vdash u?(X).P\\ \Gamma \vdash *u?(X).P\end{array}}$$

(t-nw)
$$\frac{\Gamma + \tilde{n}{:}\tilde{\mathtt{T}} \vdash P}{\Gamma \vdash (\nu\, \tilde{n} : \tilde{\mathtt{T}})P}$$

(t-cond)
$$\frac{\begin{array}{l}\Gamma \vdash u : \mathtt{T},\ v : \mathtt{T}\\ \Gamma \vdash P,\ Q\end{array}}{\Gamma \vdash \mathsf{if}\ u{=}v.P\lceil Q\rceil}$$

(t-fork)
$$\frac{\Gamma \vdash P,\ Q}{\Gamma \vdash P|Q}$$

(t-axiom)
$$\frac{}{\Gamma \vdash \mathbf{0},\ \mathsf{kill}}$$

(t-go)
$$\frac{\begin{array}{l}\Gamma \vdash u : \mathtt{loc}\\ \Gamma \vdash P\end{array}}{\Gamma \vdash \mathsf{go}\ u.P}$$

(t-ping)
$$\frac{\begin{array}{l}\Gamma \vdash u : \mathtt{loc}\\ \Gamma \vdash P,\ Q\end{array}}{\Gamma \vdash \mathsf{ping}\ u.P\lceil Q\rceil}$$

Observers

(t-obs)
$$\frac{\mathbf{pub}(\Gamma) \vdash O}{\Gamma \vdash_{\mathsf{obs}} O}$$

typed network representation, Γ. We note that reductions may still alter the network representation when a kill process is executed, yielding $\Gamma - l$, which is defined in similar fashion to the previous Chapters. Based on this definition of reduction, we are able to formalise the intuition given earlier in § 4.1 as to what reductions we want to prohibit. As usual, our discussion is guided by the following touchstone examples.

Example 4.2.4. Consider the system

$$\mathsf{err1} \Leftarrow l[\![a!\langle k\rangle.P]\!]$$

If we subject err1 to the typed network Γ, where l and a are public and k is confined, then $\Gamma \triangleright \mathsf{err1}$ breaks the boundary condition because the communicated value, k, may be inputted by an observer; if we consider the valid observer

$$O \Leftarrow l[\![a?(x).Q(x)]\!] \qquad \text{where } \Gamma \vdash_{\mathsf{obs}} l[\![Q(x)]\!]$$

then we may have the reduction

$$\Gamma \triangleright \mathsf{err1}|O \longrightarrow \Gamma \triangleright l[\![P]\!] \,|\, l[\![Q(k)]\!]$$

where the observer gains access to the confined value k, which invalidates the observer because $\Gamma \nvdash_{\mathsf{obs}} l[\![Q(k)]\!]$. If we however assume the typed network Γ to be well-formed,

Table 17. *Runtime Error*

(r-err)

$$\frac{M \;\equiv\; M'|l[\![a!\langle V\rangle.P]\!] \quad \Gamma \vdash l, a : \mathsf{p} \quad \exists v_i \in V \text{ such that } \Gamma \vdash v_i : \mathsf{c}}{\Gamma \triangleright M \longrightarrow_{\mathbf{err}}}$$

then from the assumption that a is a public channel we conclude that its object type must be public as well since well-formed types cannot have public channels which are not purely public. Stated otherwise, we should have the following judgement

$$\Gamma \vdash a : \mathsf{p}\langle\mathsf{P}\rangle$$

and as a result, according to the typing rule (t-out), we can *never* typecheck $\Gamma \vdash$ err1 which means that $\Pi \triangleright$ err1 cannot be a well-formed configuration. ■

The rule (r-err) in Table 17 formalises this notion of a *erroneous reduction step*, or runtime error. Using this formalisation, we can now prove that our type system is in some sense correct, since it rejects systems that perform error reductions.

Proposition 4.2.5 (Type Safety). If a configuration is well-formed, it cannot perform an error. Stated otherwise,

$$\Gamma \triangleright N \longrightarrow_{\mathbf{err}} \quad \text{implies} \quad \Gamma \nvdash N$$

Proof. By (r-err) of Table 17, we know:

$$N \equiv N'|l[\![a!\langle V\rangle.P]\!] \tag{4.1}$$

$$\Gamma \vdash l, a : \mathsf{p} \tag{4.2}$$

$$\exists v_i \in V \,.\, \Gamma \vdash v_i : \mathsf{c} \tag{4.3}$$

From (4.1) we deduce that, to prove $\Gamma \vdash N$, we have to use (t-par) and (t-proc), and ultimately prove:

$$\Gamma \vdash a!\langle V\rangle.P \tag{4.4}$$

But according to (4.2) and the assumption that Γ is well formed, we derive $\Gamma \vdash a : \mathsf{p}\langle\tilde{\mathsf{p}}\rangle$. From (4.3), we know that the type of V can never match the object type $\tilde{\mathsf{p}}$, and thus (t-out) can never be applied to typecheck (4.4). Thus we derive $\Gamma \nvdash N$ as required. □

Even though a configuration may not produce an error immediately, it may reduce to another configuration that may produce an error, as we see in the following examples.

Example 4.2.6. One of the reasons why we typecheck scoped names as well as free names is that a runtime error may occur as a result of scope extrusion. The type system therefore statically determines which scoped names may be scope extruded, that is the public ones, and which cannot, that is the confined ones. Consider the system

$$\mathsf{err2} \;\Leftarrow\; (\nu\, b : \mathsf{ch}\langle\mathsf{T}\rangle) l[\![a!\langle b\rangle]\!] \mid l[\![b!\langle k\rangle.P]\!]$$

If we subject this system to the typed network Γ where l and a are public and k is confined, it may result in a boundary violation irrespective of the boundary level associated with the type of the scoped channel b. Consider the valid observer

$$O \Leftarrow l[\![a?(x).x?(y).Q(y)]\!]$$

If the type associated with b it is confined, that is $\mathtt{ch}\langle \mathtt{T}\rangle = \mathtt{c}\langle \mathtt{T}\rangle$, then clearly after the first reduction

$$\Gamma \triangleright \mathsf{err2} \,|\, O \;\longrightarrow\; \Gamma \triangleright (\nu\, b\!:\!\mathtt{c}\langle \mathtt{T}\rangle)(l[\![b!\langle k\rangle.P]\!] \,|\, l[\![b?(y).Q(y)]\!])$$

the observer $l[\![b?(y).Q(y)]\!]$ has gained access to the confined (scoped) name b. If, on the other hand, b is public, that is $\mathtt{ch}\langle \mathtt{T}\rangle = \mathtt{p}\langle \mathtt{T}\rangle$, then the observer gains access to a confined value, in this case k, by the second reduction, $l[\![Q(k)]\!]$, after the scope extrusion of b

$$\Gamma \triangleright \mathsf{err2} \,|\, O \;\longrightarrow\; \cdot \;\longrightarrow \Gamma \triangleright (\nu\, b\!:\!\mathtt{p}\langle \mathtt{T}\rangle)(l[\![P]\!] \,|\, l[\![Q(k)]\!])$$

It turns out that the typing rules of Table 16 *do not typecheck* err2 for either case of the type $\mathtt{ch}\langle \mathtt{T}\rangle$:

- If $\mathtt{ch}\langle \mathtt{T}\rangle = \mathtt{c}\langle \mathtt{T}\rangle$, then we are not able to typecheck $l[\![a!\langle b\rangle]\!]$: since the type of a is (purely) public, its object type is also public and can therefore never be matched by the type of b in (t-out), which is confined, $\mathtt{c}\langle \mathtt{T}\rangle$.
- If $\mathtt{ch}\langle \mathtt{T}\rangle = \mathtt{p}\langle \mathtt{T}\rangle$, then we also conclude, by the same restriction on public channel types, that the object type $\mathtt{T}$, must be public. As a result, we cannot typecheck $l[\![b!\langle k\rangle.P]\!]$ using (t-out) because we cannot match this public object type with the type of k which we assumed to be confined.

Even without scope extrusion, it can be non-trivial to statically determine whether a system may eventually produce a runtime error. The following example illustrates why we make use of object types to typecheck channels. Consider the following system

$$\mathsf{err3} \;\Leftarrow\; l[\![b!\langle k\rangle]\!] \,|\, l[\![b?(x).a!\langle x\rangle.P]\!]$$

subject to the typed network representation Γ, where l and a are public and k and b are confined names. Even though the output of the confined location k is a valid one, since it happens on a confined channel b, the configuration $\Gamma \triangleright$ err3 may produce an error after one internal reduction because

$$\Gamma \triangleright \mathsf{err3} \;\longrightarrow\; \Gamma \triangleright l[\![a!\langle k\rangle.P]\!] \longrightarrow_{\mathbf{err}}$$

However, to be able to reject err3, the type system needs to use channel object types. Thus

- if $\Gamma \vdash b\!:\!\mathtt{c}\langle \mathsf{c}[]\rangle$ we reject err3 on the basis that, from b we should expect a confined location, $\mathtt{c}[-]$: the rule (t-in-rep) typechecks $a!\langle x\rangle.P$ under the assumption that x has the confined type $\mathtt{c}[-]$ and since the type of a is purely public, the rule (t-out) rejects it.
- If, on the other hand, $\Gamma \vdash b\!:\!\mathtt{c}\langle \mathtt{p}[-]\rangle$, we still reject err3 since (t-out) does not typecheck the process $b!\langle k\rangle$ on the premise that the type of the value outputted k, that is $\mathtt{c}[-]$, does not match the object type of the channel, $\mathtt{p}[-]$. ■

Example 4.2.6 shows how the type system rejects a number of systems that may produce runtime errors indirectly, either as a result of a sequence of interaction with the observer (err2), or after a sequence of internal reductions (err3). We however need to prove that the type system rejects any such system that may produce a runtime error. Stated otherwise, from Lemma 4.2.5 we know that well-formed configurations do not produce an error immediatley; by showing that our reduction system is closed under well-formed configurations, we can conclude that a well-formed configuration can never reduce to another configuration that can produce an error.

The proposition stating closure over well-formed configurations is called Subject Reduction ; this Proposition makes us of the following lemmas, the most notable of which is Lemma 4.2.10, the Substitution Lemma, defined for processes instead of system.

Lemma 4.2.7. If $\Gamma \vdash M$ and $M \equiv N$, then $\Gamma \vdash N$.

Proof. Proof by induction on the length of derivation on why $M \equiv N$. □

Lemma 4.2.8 (Weakening). If $\Gamma \vdash M$ and $\Gamma + u:\mathtt{T}$ is a valid well formed network representation, then $\Gamma + u:\mathtt{T} \vdash M$

Proof. Proof by induction on the derivation of $\Gamma \vdash M$ □

Lemma 4.2.9 (Strenghtening). If $\Gamma + u:\mathtt{T} \vdash M$ and $n \notin (\mathbf{fn}(M) \cup \mathbf{fv}(M))$, then $\Gamma \vdash M$.

Proof. Proof by induction on the derivation of $\Gamma + u:\mathtt{T} \vdash M$ □

Lemma 4.2.10 (Substitution). If $\Gamma + x:\mathtt{T} \vdash P$ and $\Gamma \vdash v:\mathtt{T}$, then $\Gamma \vdash P\{v/x\}$

Proof. The proof progresses by induction on the length of derivation of $\Gamma + x:\mathtt{T} \vdash P$. In this proof, whenever $x \notin \mathbf{fv}(P)$, then $P\{v/x\} = P$ and as a consequence, the result follows from Lemma 4.2.9 (Strenghtening). We thus focus on the cases where $x \in \mathbf{fv}(P)$.

Case the last rule used was:

- (t-axiom), this would constitutes the base case. It is immediately true by the fact that $x \notin \mathbf{fv}(\mathbf{0}, \mathsf{kill})$.
- (t-out), we know $P \equiv u!\langle V\rangle.Q$, $\Gamma \vdash u:\mathbf{ch}\langle\tilde{\mathtt{U}}\rangle$ $\Gamma \vdash V:\tilde{\mathtt{U}}$ and $\Gamma \vdash Q$.
  - If $x = u$ then $\Gamma \vdash v:\mathbf{ch}\langle\tilde{\mathtt{U}}\rangle$
  - If $x = v_i \in V$ then $\Gamma \vdash V\{v/x\}:\tilde{\mathtt{U}}$
  - If $x \in \mathbf{fv}(Q)$ then by inductive hypothesis we obtain $\Gamma \vdash Q\{v/x\}$

  As a result, in any of the above cases, we are still able to reconstruct $\Gamma \vdash (u!\langle V\rangle.Q)\{v/x\}$.
- (t-in-rep), we know $P \equiv u?(X).Q$ or $P \equiv *u?(X).Q$, $\Gamma \vdash u : \mathbf{ch}\langle\tilde{\mathtt{U}}\rangle$ and $\Gamma + X:\mathtt{T} \vdash Q$
  - If $x = u$ then we know that $\Gamma \vdash v : \mathbf{ch}\langle\tilde{\mathtt{U}}\rangle$
  - If $x \in \mathbf{fv}(Q)$ then by inductive hypothesis we obtain $\Gamma \vdash Q\{v/x\}$

  As a result, in either case we are still able to reconstruct $\Gamma \vdash (u?(X).Q)\{v/x\}$ and $\Gamma \vdash (*u?(X).Q)\{v/x\}$

- (t-ping), we know $P \equiv \text{ping}\ u.P\lceil Q \rceil,\ \Gamma \vdash u\!:\!\texttt{loc},\ \Gamma \vdash P, Q$.
  - If $x = u$ then we know $\Gamma \vdash v : \texttt{loc}$
  - By inductive hypothesis we obtain $\Gamma \vdash P\{^v\!/\!_x\},\ Q\{^v\!/\!_x\}$

  As a result, we are always able to reconstruct $\Gamma \vdash (\text{ping}\ u.P\lceil Q \rceil)\{^v\!/\!_x\}$

- (t-match), (t-fork), (t-go) and (t-new) are similar to the cases above. □

We are now in a position to prove one of the main (standard) results of our type system, Subject Reduction.

Proposition 4.2.11 (Subject Reduction). If $\Gamma \vdash M$ and $\Gamma \triangleright M \longrightarrow \Gamma' \triangleright N$, then $\Gamma \vdash N$

Proof. The proof proceed by induction on the derivation of $\Gamma \triangleright M \longrightarrow \Gamma' \triangleright N$. We here analyse in depth the most involving cases and leave the remainder to the interested reader.

Case the last rule used to derive $\Gamma \triangleright M \longrightarrow \Gamma' \triangleright N$:

- (r-comm), we know $M \equiv l[\![a!\langle V \rangle.P]\!] | l[\![a?(X).Q]\!],\ N \equiv l[\![P]\!] | l[\![Q\{^V\!/\!_X\}]\!]$ and $\Gamma = \Gamma'$. From $\Gamma \vdash M$, (t-par) and (t-proc) we derive

$$\Gamma \vdash l : \texttt{loc} \tag{4.5}$$
$$\Gamma \vdash a!\langle V \rangle.P \tag{4.6}$$
$$\Gamma \vdash a?(X).Q \tag{4.7}$$

  From (4.7) and (t-in-rep) we obtain

$$\Gamma \vdash a : \text{ch}\langle \texttt{T} \rangle \tag{4.8}$$
$$\Gamma + X\!:\!\texttt{T} \vdash Q \tag{4.9}$$

  Moreover, from (4.6) and the fact that names have unique types (Lemma 4.2.2) we derive

$$\Gamma \vdash V\!:\!\texttt{T} \tag{4.10}$$
$$\Gamma \vdash P \tag{4.11}$$

  From (4.9), (4.10) and Substitution Lemma 4.2.10 we obtain

$$\Gamma \vdash Q\{^V\!/\!_X\} \tag{4.12}$$

  and using (t-proc), (4.5), (4.11), (4.12) and finally (t-par), we are able to reconstruct $\Gamma \vdash l[\![P]\!] | l[\![Q\{^V\!/\!_X\}]\!]$.

- (r-rep) we know $M \equiv l[\![*a?(X).P]\!],\ N \equiv l[\![a?(X).(P| * a?(Y).P\{^Y\!/\!_X\})]\!]$ and $\Gamma' = \Gamma$. From $\Gamma \vdash l[\![*a?(X).P]\!]$, (t-proc), (t-in-rep) we obtain

$$\Gamma \vdash l : \texttt{loc} \tag{4.13}$$
$$\Gamma \vdash a : \text{ch}\langle \texttt{T} \rangle \tag{4.14}$$
$$\Gamma + X\!:\!\texttt{T} \vdash P \tag{4.15}$$

  from (4.15), the Weakening Lemma 4.2.8 and the Substitution Lemma 4.2.10, we obtain

$$\Gamma + X\!:\!\texttt{T} + Y\!:\!\texttt{T} \vdash P\{^Y\!/\!_X\} \tag{4.16}$$

and from (4.16), (4.14) and (t-in-rep) we derive

$$\Gamma + X{:}\mathtt{T} \vdash *a?(Y).P\{^{Y}\!/\!_{X}\} \tag{4.17}$$

Subsequently, by (t-fork), (4.15), (4.17), (4.14) and (t-in-rep) we obtain

$$\Gamma \vdash a?(X).(P \mid *\, a?(Y).P\{^{Y}\!/\!_{X}\}) \tag{4.18}$$

and thus from (4.13) and (t-proc) $\Gamma \vdash l[\![a?(X).(P| *\, a?(Y).P\{^{Y}\!/\!_{X}\})]\!]$ as required.

- (r-new) we know $M \equiv l[\![(\nu n{:}\mathtt{T})P]\!]$, $N \equiv (\nu\, n : \mathtt{T})l[\![P]\!]$ and $\Gamma = \Gamma'$. From $\Gamma \vdash l[\![(\nu n : \mathtt{T})P]\!]$, (t-proc) and (t-new) we get

$$\Gamma \vdash l : \mathtt{loc} \tag{4.19}$$
$$\Gamma + n{:}\mathtt{T} \vdash P \tag{4.20}$$

By weakening and (4.19) we get

$$\Gamma + n : \mathtt{T} \vdash l : \mathtt{loc} \tag{4.21}$$

Thus by (4.20), (4.21) and (t-proc) we derive $\Gamma + n : \mathtt{T} \vdash l[\![P]\!]$ and by (t-rest) we get $\Gamma \vdash (\nu\, n{:}\mathtt{T})l[\![P]\!]$ as required. □

4.2.2 Equivalences in typed DπLoc

As a touchstone equivalence for comparing typed DπLoc configurations, we use reduction barbed congruence once again. We carry over most definitions from Chapter 2, with the exception of those dealing with observables, which we now state.

Definition 4.2.12 (Barbs for Typed DπLoc). $\Gamma \triangleright N \Downarrow_{a@l}$ denotes an *observable barb* exhibited by the configuration $\Gamma \triangleright N$, on channel a at location l. Formally, it means that $\Gamma \triangleright N \longrightarrow^{*} \Gamma' \triangleright N'$ for some $\Gamma' \triangleright N'$ such that $N' \equiv (\nu\, \tilde{n}{:}\tilde{\mathtt{T}})M|l[\![a!\langle V\rangle.Q]\!]$, $\Gamma \vdash l{:}\mathbf{alive}$ and $\Gamma \vdash_{\mathsf{obs}} l, a$ ■

Definition 4.2.12 strengthens the previous definition of a barb, requiring in addition, that the channel and location of the barb are *public*, that is $\Gamma \vdash_{\mathsf{obs}} l, a$. This further requirement stems from the fact that valid observers are limited to public resources and well-formed configuration never let observers gain access to confined names. In fact the boundary information, vested as the stateless types and the respective type system, is only required *internally* by the configuration, in order to ensure that the configuration does not violate the restricted view. Stated otherwise, the observer should be oblivious to confined boundary types since it uses only public names. Moreover, we can also claim that observers are oblivious to boundary tags altogether since it uses only one kind. We therefore define the partial function over types

$$\mathsf{simp}(\mathtt{T}) \stackrel{\text{def}}{=} \begin{cases} \mathtt{loc[A]} & \text{if } \mathtt{T} = \mathtt{loc[p,A]} \\ \mathtt{ch} & \text{if } \mathtt{T} = \mathtt{ch}\langle\mathtt{p},\tilde{\mathtt{W}}\rangle \end{cases}$$

which allows us to recover the simple state types used in Chapter 2. In the sequel, we allow the type variables S, R to range over these simple state types loc[A] and ch. This operation allows us to recover the previous definition of an observer of Chapter 2, with the interpretation that previous observers defined in terms of simple state types used

public names only. It also allows us to recover the observer's knowledge of the state of the network used in Chapter 2: more specifically, the observer is only concerned with the knowledge regarding public names and the state of public locations. We formalise this in the next definition.

Definition 4.2.13 (Knowledge Representation). As before, we use $\mathcal{I}$ to range over *knowledge representations* in typed DπLoc. It consisting of pairs $\langle \mathcal{N}, \mathcal{O} \rangle$ where:

- $\mathcal{N}$ is a set of names, $\mathcal{N} \subseteq \textsc{Names}$.
- $\mathcal{O}$ is a set of *public* location names from $\mathcal{N}$ that are alive, $\mathcal{O} \subseteq \mathbf{loc}(\mathcal{N})$.

The knowledge representation can be obtained from a typed network representation as:

$$\mathcal{I}(\Gamma) \stackrel{\text{def}}{=} \langle \{n \mid \Gamma \vdash_{\mathsf{obs}} n\},\ (\Gamma_{\mathcal{A}} \cap \{l \mid \Gamma \vdash_{\mathsf{obs}} l\}) \rangle$$

where we filter the public names from $\Gamma_{\mathcal{T}}$ and strip away their type and also filter away dead confined locations from $\mathcal{A}$. ■

When we give our fault tolerance definitions in § 4.3, we find it useful to define an alternative form of contextual relation, used to study systems under the assumption that the state of the underlying (public) network is static and does not suffer further faults. For this we need contexts that do not induce faults.

Definition 4.2.14 (Failure-Free Contextual typed relations). A typed relation $\mathcal{R}$ over configurations is *failure-free contextual* if:

(Failure-free Parallel Systems)

- $\Gamma \models M \,\mathcal{R}\, N$ and $\Gamma \vdash_{\mathsf{obs}} O$, where O does not use kill implies
  - $\Gamma \models M|O \,\mathcal{R}\, N|O$
  - $\Gamma \models O|M \,\mathcal{R}\, O|N$

(Network Extensions)

- $\Gamma \models M \,\mathcal{R}\, N$ and $\Gamma \vdash_{\mathsf{obs}} \mathtt{T}$, n fresh implies $\Gamma + n : \mathtt{T} \models M \,\mathcal{R}\, N$

■

We thus inherit the definitions of reduction closed, barb preserving and contextual relations from Chapter 2 and that of typed relations from Chapter 3 and together with Definition 4.2.14, we define two reduction barbed congruences.

Definition 4.2.15 (Reduction Barbed Congruences for typed DπLoc).

- *Reduction barbed congruence*, denoted by $\cong$, is the largest symmetric typed relation over configurations which is:
  - barb preserving
  - reduction closed
  - contextual

- *Failure-free reduction barbed congruence*, denoted by $\cong_{ff}$, is the largest symmetric typed relation over configurations which is:
  - barb preserving
  - reduction closed
  - *failure-free* contextual

Both congruences are once again denoted as

$$\mathcal{I} \models \Gamma_1 \triangleright M_1 \cong \Gamma_2 \triangleright M_2 \quad \text{whenever} \quad \mathcal{I}(\Gamma_1) = \mathcal{I}(\Gamma_2)$$

and we specialise the notation to

$$\Gamma \models M \cong N$$

when we are simply comparing systems running on the same network, that is $\mathcal{I}(\Gamma) \models \Gamma \triangleright M \cong \Gamma \triangleright N$. ■

The latter reduction barbed congruence, $\cong_{ff}$, is a weaker form or congruence, where two systems are evaluated under the assumption that the underlying observable network will not incur further failure; since we use this relation later on for our definition of fault tolerance, we defer any further discussion about it to § 4.3.

Once again, both reduction barbed congruence definitions suffer form the problem of quantification over all contexts, discussed earlier in Chapters 2 and 3; this makes them harder to use in practice to show that two configurations are equivalent. We solve this problem as before, by defining a *derived* lts for two bisimulations that are sound and complete with respect to these equivalence. This derived lts simply filters out boundary information from the types of bound names present in transition labels using the operation $\mathsf{simp}(\mathtt{T})$ defined earlier; derived labels thus use only simple types. We start by presenting the full lts.

Definition 4.2.16 (Labelled Transition System for typed DπLoc). The lts for typed DπLoc consists of the collection of actions, $\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'$, where μ can take one of the forms

- τ (internal action)
- $(\tilde{n} : \tilde{\mathtt{T}})l : a?(V)$ (bound input)
- $(\tilde{n} : \tilde{\mathtt{T}})l : a!\langle V\rangle$ (bound output)
- $\mathsf{kill} : l$ (external location killing)

The transitions, are defined as the least relations satisfying different axioms and rules, found in Tables 18, 19 and 20. ■

The main difference between the rules in Tables 18, 19 and 20 and those given in Chapter 2 are those dealing with external actions, namely (l-out), (l-in), (l-halt). These axioms introduce extra side conditions for actions denoting interactions to and from the observer, requiring that they involve *public* channels and locations only.

Table 18. *Local Operational Rules for Typed DπLoc*

Assuming $\Gamma \vdash l : \textbf{alive}$

(l-in)

$$\frac{}{\Gamma \triangleright l[\![a?(X).P]\!] \xrightarrow{l:a?(V)} \Gamma \triangleright l[\![P\{^{V}\!/\!X\}]\!]}\ \Gamma \vdash_{\mathsf{obs}} l,\ \Gamma \vdash a : \mathtt{p}\langle\tilde{\mathtt{W}}\rangle,\ V : \tilde{\mathtt{W}}$$

(l-out)

$$\frac{}{\Gamma \triangleright l[\![a!\langle V\rangle.P]\!] \xrightarrow{l:a!\langle V\rangle} \Gamma \triangleright l[\![P]\!]}\ \Gamma \vdash_{\mathsf{obs}} l, a$$

(l-fork)

$$\frac{}{\Gamma \triangleright l[\![P|Q]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(l-in-rep)

$$\frac{}{\Gamma \triangleright l[\![*a?(X).P]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![a?(X).(P| *a?(Y).P\{^{Y}\!/\!X\})]\!]}$$

(l-eq)

$$\frac{}{\Gamma \triangleright l[\![\mathsf{if}\ u = u.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![P]\!]}$$

(l-neq)

$$\frac{}{\Gamma \triangleright l[\![\mathsf{if}\ u = v.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![Q]\!]}\ u \neq v$$

Table 19. *Network Operational Ruled for typed DπLoc*

Assuming $\Gamma \vdash l : \textbf{alive}$

(l-kill)

$$\frac{}{\Gamma \triangleright l[\![\mathsf{kill}]\!] \xrightarrow{\tau} (\Gamma - l) \triangleright l[\![\mathbf{0}]\!]}$$

(l-halt)

$$\frac{}{\Gamma \triangleright N \xrightarrow{\mathsf{kill}:l} (\Gamma - l) \triangleright N}\ \Gamma \vdash_{\mathsf{obs}} l : \textbf{alive}$$

(r-new)

$$\frac{}{\Gamma \triangleright l[\![(\nu n : \mathtt{T})P]\!] \longrightarrow \Gamma \triangleright (\nu\, n : \mathtt{T})\, l[\![P]\!]}$$

(r-go)

$$\frac{}{\Gamma \triangleright l[\![\mathsf{go}\ k.P]\!] \xrightarrow{\tau} \Gamma \triangleright k[\![P]\!]}\ \Gamma \vdash k \leftarrow l$$

(r-ngo)

$$\frac{}{\Gamma \triangleright l[\![\mathsf{go}\ k.P]\!] \xrightarrow{\tau} \Gamma \triangleright k[\![\mathbf{0}]\!]}\ \Gamma \nvdash k \leftarrow l$$

(r-ping)

$$\frac{}{\Gamma \triangleright l[\![\mathsf{ping}\ k.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![P]\!]}\ \Gamma \vdash k \leftarrow l$$

(r-nping)

$$\frac{}{\Gamma \triangleright l[\![\mathsf{ping}\ k.P\lceil Q\rceil]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![Q]\!]}\ \Gamma \nvdash k \leftarrow l$$

As a result of the conditions restricting names in labels to public names only, we also modify (l-par-comm) so as to remove any of these restrictions for internal communication; the operation $\uparrow(\Gamma)$ translates all the confined types in Γ into perfectly public types, thereby collapsing the public network in Γ into the biggest possible, as in the case of (l-par-comm)

Table 20. *Context Operational Rules for typed DπLoc*

(l-open)

$$\frac{\Gamma + n : \mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Gamma' \triangleright N'}{\Gamma \triangleright (\nu\, n : \mathtt{T})N \xrightarrow{(n:\mathtt{T},\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Gamma' \triangleright N'}\ l, a \neq n \in V$$

(l-weak)

$$\frac{\Gamma + n : \mathtt{T} \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Gamma' \triangleright N'}{\Gamma \triangleright N \xrightarrow{(n:\mathtt{T},\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Gamma' \triangleright N'}\ l, a \neq n \in V$$

(l-rest)

$$\frac{\Gamma + n : \mathtt{T} \triangleright N \xrightarrow{\mu} \Gamma' + n : \mathtt{U} \triangleright N'}{\Gamma \triangleright (\nu\, n : \mathtt{T})N \xrightarrow{\mu} \Gamma' \triangleright (\nu\, n : \mathtt{U})N'}\ n \notin \mathbf{fn}(\mu)$$

(l-par-ctxt)

$$\frac{\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'}{\begin{array}{c}\Gamma \triangleright N\,|\,M \xrightarrow{\mu} \Gamma' \triangleright N'\,|\,M \\ \Gamma \triangleright M\,|\,N \xrightarrow{\mu} \Gamma' \triangleright M\,|\,N'\end{array}}\ \Gamma \vdash M$$

(l-par-comm)

$$\frac{\uparrow(\Gamma) \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a!\langle V\rangle} \Gamma' \triangleright N' \qquad \uparrow(\Gamma) \triangleright M \xrightarrow{(\tilde{n}:\tilde{\mathtt{T}})l:a?(V)} \Gamma'' \triangleright M'}{\begin{array}{c}\Gamma \triangleright N\,|\,M \xrightarrow{\tau} \Gamma \triangleright (\nu\,\tilde{n} : \tilde{\mathtt{T}})(N'\,|\,M') \\ \Gamma \triangleright M\,|\,N \xrightarrow{\tau} \Gamma \triangleright (\nu\,\tilde{n} : \tilde{\mathtt{T}})(M'\,|\,N')\end{array}}$$

of Table 13. It is defined as

$$\uparrow(\Gamma) \;=\; \langle \{n : \uparrow(\mathtt{W}) \;\; | \Gamma \vdash n : \mathtt{W}\}, \Gamma_{\mathcal{A}} \rangle$$

$$\uparrow(\mathtt{W}) \;=\; \begin{cases} \mathtt{p}\langle \uparrow(\tilde{\mathtt{W}})\rangle & \text{if } \mathtt{T} = \mathtt{ch}\langle\tilde{\mathtt{W}}\rangle \\ \mathtt{p}[] & \text{if } \mathtt{T} = \mathtt{loc} \end{cases} \qquad \text{and} \qquad \uparrow(\mathtt{W}_1, .., \mathtt{W}_n) \;=\; \uparrow(\mathtt{W}_1), .., \uparrow(\mathtt{W}_n)$$

As we stated earlier, the bisimulations are defined on a derived lts, using simple types for bound names in labels instead of the full boundary types, required only for type-checking reasons.

Definition 4.2.17 (Derived Labelled Transition System for typed DπLoc). The derived lts for typed DπLoc is similar to the untyped version, consisting of the same collection of actions, $\Gamma \triangleright N \overset{\mu}{\longmapsto} \Gamma' \triangleright N'$, where μ can take one of the forms

- τ (internal action)
- $(\tilde{n} : \tilde{\mathtt{S}})l : a?(V)$ (bound input)
- $(\tilde{n} : \tilde{\mathtt{S}})l : a!\langle V\rangle$ (bound output)
- kill : l (external location killing)

The transitions, are defined as the least relations satisfying different axioms and rules, found in Tables 18, 19 and 20 just presented and the filtering rules in Table 21. ■

Based on this derived lts, we now define the required bisimulations. While we inherit the standard bisimulation definition from Definition 2.3.2 for the general bisimulation

Table 21. *The derived lts for typed DπLoc*

(l-deriv-1)
$$\frac{\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'}{\Gamma \triangleright N \overset{\mu}{\longmapsto} \Gamma' \triangleright N'} \; \mu \in \{\tau, \mathsf{kill} : l\}$$

(l-deriv-2)
$$\frac{\Gamma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Gamma' \triangleright N'}{\Gamma \triangleright N \overset{(\tilde{n}:\mathsf{simp}(\tilde{\mathrm{T}}))l:a!\langle V\rangle}{\longmapsto} \Gamma' \triangleright N'}$$

(l-deriv-3)
$$\frac{\Gamma \triangleright N \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Gamma' \triangleright N'}{\Gamma \triangleright N \overset{(\tilde{n}:\mathsf{simp}(\tilde{\mathrm{T}}))l:a?(V)}{\longmapsto} \Gamma' \triangleright N'}$$

ranging over all the labels in Definition 4.2.17, we need to come up with an alternative definition for a bisimulation that corresponds to failure-free reduction barbed congruence: we do this by restricting the range of actions.

Definition 4.2.18 (Failure-free weak bisimulation equivalence). We let γ range over the actions $(\tilde{n} : \tilde{\mathrm{S}})l : a!\langle V\rangle$, $(\tilde{n} : \tilde{\mathrm{S}})l : a?(V)$ and τ but not $\mathsf{kill} : l$. Then failure-free weak bisimulation equivalence, denoted as $\approx_{\mathit{ff}}$, is defined to be the largest typed relation over configurations such that if $\Gamma_1 \triangleright M_1 \approx_{\mathit{ff}} \Gamma_2 \triangleright M_2$ then

- $\Gamma_1 \triangleright M_1 \overset{\gamma}{\longmapsto} \Gamma_1' \triangleright M_1'$ implies $\Gamma_2 \triangleright M_2 \overset{\hat{\gamma}}{\Longmapsto} \Gamma_2' \triangleright M_2'$ such that $\Gamma_1' \triangleright M_1' \approx_{\mathit{ff}} \Gamma_2' \triangleright M_2'$
- $\Gamma_2 \triangleright M_2 \overset{\gamma}{\longmapsto} \Gamma_2' \triangleright M_2'$ implies $\Gamma_1 \triangleright M_1 \overset{\hat{\gamma}}{\Longmapsto} \Gamma_1' \triangleright M_1'$ such that $\Gamma_1' \triangleright M_1' \approx_{\mathit{ff}} \Gamma_2' \triangleright M_2'$ ■

It is easy to see that failure free weak bisimulation equivalence is *weaker* than the standard weak bisimulation equivalence, as we show below.

Proposition 4.2.19. If $\Gamma_1 \triangleright M_1 \approx \Gamma_2 \triangleright M_2$ then $\Gamma_1 \triangleright M_1 \approx_{\mathit{ff}} \Gamma_2 \triangleright M_2$.

Proof. We define $\mathcal{R}$ as

$$\mathcal{R} \;=\; \left\{ \Gamma_1 \triangleright M_1 \quad, \Gamma_2 \triangleright M_2 \qquad \middle| \; \Gamma_1 \triangleright M_1 \approx \Gamma_2 \triangleright M_2 \right\}$$

and show that $\mathcal{R} \subseteq \approx_{\mathit{ff}}$. The result follows since the actions over which $\approx_{\mathit{ff}}$ is defined are a subset of the actions over which $\approx$ is defined. □

Just as in Chapters 2 and 3 we need to justify the use of bisimulations for relating systems, based on the equivalences defined in Definition 4.2.15. First of all, we need to show that the lts is closed with respect to well-formed configurations. We here do not give the full proof, since this is similar to Proposition 3.5.13.

Similar to the proofs in Chapter 3, the proof of soundness for both $\approx$ and $\approx_{\mathit{ff}}$ hinges on whether they are contextual relations or not. The proof for contextuality in turn uses extensively the Composition and Decomposition Lemmas, which have to be set up specifically for the lts of typed DπLoc. We here state these three main results without

delving into the specifics of the proofs, since they turn out to be similar to the proofs given in Chapter 3. We only note that the clauses dealing with composing and decomposing internal communication in Lemmas 4.2.20 and 4.2.21 specify that the types of both the bound inputs and the bound outputs match: this is because we know that the channel on which the communication happens has to be public, and since we assume well-formed configurations that used only perfectly public types, we know that the respective object type is public as well.

Lemma 4.2.20 (Composition).

- Suppose $\Gamma \triangleright M \overset{\mu}{\longmapsto} \Gamma' \triangleright M'$. If $\Gamma \vdash N$ for arbitrary system N, then
  - $\Gamma \triangleright M|N \overset{\mu}{\longmapsto} \Gamma' \triangleright M'|N$
  - $\Gamma \triangleright N|M \overset{\mu}{\longmapsto} \Gamma \triangleright N|M$
- Suppose $\Gamma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Gamma' \triangleright M'$ and $\Gamma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Gamma'' \triangleright N'$. Then
  - $\Gamma \triangleright M|N \overset{\tau}{\longmapsto} \Gamma \triangleright (\nu\, \tilde{n}:\tilde{\mathrm{T}})M'|N'$
  - $\Gamma \triangleright N|M \overset{\tau}{\longmapsto} \Gamma \triangleright (\nu\, \tilde{n}:\tilde{\mathrm{T}})N'|M'$

Proof. (Outline) The proof progresses by extracting the necessary structure of the systems M, N and the network Γ to be able to re-compose them using rules such as (l-par-ctxt), (l-par-comm) and (l-rest) □

Lemma 4.2.21 (Decomposition). Suppose $\Gamma \triangleright M|N \overset{\mu}{\longmapsto} \Gamma' \triangleright M'$ where $\Gamma \vdash_{\mathsf{obs}} M$ or $\Gamma \vdash_{\mathsf{obs}} N$. Then, one of the following conditions hold:

1. M' is $M''|N$, where $\Gamma \triangleright M \overset{\mu}{\longmapsto} \Gamma' \triangleright M''$.
2. M' is $M|N'$ and $\Gamma \triangleright N \overset{\mu}{\longmapsto} \Gamma' \triangleright N'$.
3. M' is $(\nu\, \tilde{n}:\tilde{\mathrm{T}})M''|N'$, μ is τ, $\Gamma' = \Gamma$ and either
   - $\Gamma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Gamma'' \triangleright M''$ and $\Gamma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Gamma''' \triangleright N'$
   - $\Gamma \triangleright M \xmapsto{(\tilde{n}:\tilde{\mathrm{T}})l:a?(V)} \Gamma'' \triangleright M''$ and $\Gamma \triangleright N \xmapsto{(\tilde{n}:\tilde{\mathrm{T}})l:a!\langle V\rangle} \Gamma''' \triangleright N'$

Proof. (Outline) The proof progressed by induction on the derivation of $\Gamma \triangleright M|N \overset{\mu}{\longmapsto} \Gamma' \triangleright M'$. □

Proposition 4.2.22 (Contextuality of Behavioural Equivalence).

- If two configurations are bisimilar, they are also bisimilar under any context. Stated otherwise, $\mathcal{I} \models \Gamma_1 \triangleright M_1 \;\approx\; \Gamma_2 \triangleright M_2$ implies that for $\mathcal{I} \vdash_{\mathsf{obs}} O, \mathtt{T}$ and n fresh in $\mathcal{I}$ we have:
  - $\Gamma_1 \triangleright M_1|O \approx \Gamma_2 \triangleright M_2|O$ and $\Gamma_1 \triangleright O|M_1 \approx \Gamma_2 \triangleright O|M_2$
  - $\Gamma_1{+}n:\mathtt{T} \triangleright M_1 \approx \Gamma_2{+}n:\mathtt{T} \triangleright M_2$

- If two configurations are failure-free bisimilar, they are also failure-free bisimilar under any context. Stated otherwise, $\mathcal{I} \models \Gamma_1 \triangleright M_1 \approx_{ff} \Gamma_2 \triangleright M_2$ implies that for $\mathcal{I} \vdash_{\mathsf{obs}} O, \mathtt{T}$ where $O \not\equiv O|l[\![\mathsf{kill}]\!]$ and n fresh in $\mathcal{I}$ we have:
  - $\Gamma_1 \triangleright M_1|O \approx_{ff} \Gamma_2 \triangleright M_2|O$ and $\Gamma_1 \triangleright O|M_1 \approx_{ff} \Gamma_2 \triangleright O|M_2$
  - $\Gamma_1{+}n{:}\mathtt{T} \triangleright M_1 \approx_{ff} \Gamma_2{+}n{:}\mathtt{T} \triangleright M_2$

Proof. (Outline) The proof for both clauses are similar. For example, the proof for the first clause proceeds by inductively defining a relation $\mathcal{R}$ as the largest typed relation over configurations satisfying:

$$\mathcal{R} = \left\{ \begin{array}{ll} \langle \Gamma_1 \triangleright M_1,\ \Gamma_2 \triangleright M_2 \rangle & \mid \Gamma_1 \triangleright M_1 \approx \Gamma_2 \triangleright M_2 \\[2ex] \begin{array}{l} \langle \Gamma_1 \triangleright M_1|O,\ \Gamma_2 \triangleright M_2|O \rangle \\ \langle \Gamma_1 \triangleright O|M_1,\ \Gamma_2 \triangleright O|M_2 \rangle \end{array} & \left| \ \Gamma_1 \triangleright M_1 \mathcal{R} \Gamma_2 \triangleright M_2 \right. \\[2ex] \langle \Gamma_1{+}n{:}\mathtt{T} \triangleright M_1|O,\ \Gamma_2{+}n{:}\mathtt{T} \triangleright M_2|O \rangle & \left| \begin{array}{l} \mathcal{I} \models \Gamma_1 \triangleright M_1 \ \mathcal{R}\ \Gamma_2 \triangleright M_2, \\ \mathcal{I} \vdash \mathtt{T} \text{ and } n \text{ is fresh} \end{array} \right. \\[2ex] \langle \Gamma_1 \triangleright (\nu\, n{:}\mathtt{T})M_1,\ \Gamma_2 \triangleright (\nu\, n{:}\mathtt{U})M_2 \rangle & \mid \Gamma_1{+}n{:}\mathtt{T} \triangleright M_1 \mathcal{R} \Gamma_2{+}n{:}\mathtt{U} \triangleright M_2 \end{array} \right\}$$

and showing that $\mathcal{R} \subseteq \approx$; since $\approx$ is the biggest possible relation, this would mean that it is contextual. □

The proof of completeness relies on the concept of definability, where we have to show, amongst other things, that the observer is capable of determining the current state of the public locations as well as the state of the scope extruded locations; this task is facilitated by the fact that well-formed configurations scope extrude public locations only.

Lemma 4.2.23 (Observable Network). Assume that for arbitrary network representation Γ, Γ_+ stands for

$$\Gamma + k_0{:}\mathtt{p[a]} + \textsc{succ}{:}\mathtt{p}\langle - \rangle$$

Then, for every $\mathcal{I} = \langle \mathcal{N}, \mathcal{O} \rangle$ and every Γ, there exists a system $N^{\mathcal{I}}$ where $\Gamma_+ \vdash N^{\mathcal{I}}$ such that,

$$\Gamma_+ \triangleright N^{\mathcal{I}} \longrightarrow^* \Gamma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] \text{ iff } \mathcal{O} = \mathcal{I}(\Gamma)_{\mathcal{O}} \text{ and} \mathbf{loc}(\mathcal{N}) = \mathbf{loc}(\mathbf{pub}(\Gamma)_{\mathcal{N}}).$$

Stated otherwise, $N^{\mathcal{I}}$ can determine the state of every public location.

Proof. (Outline) We set the process $N^{\mathcal{I}}$ to $k_0[\![\mathit{verPubStat}^{\mathcal{I}}(\textsc{succ})]\!]$, where $\mathit{verPubStat}^{\mathcal{I}}(x)$ is defined as:

$$\mathit{verPubStat}^{\langle \mathcal{N}, \mathcal{O} \rangle}(x) \Leftarrow (\nu\, \mathit{sync}) \left(\begin{array}{l} \prod_{l \in \mathcal{O}} \mathsf{ping}\ l.\lceil \mathit{sync}!\langle\rangle \rceil \mid \prod_{l \in (\mathbf{loc}(\mathcal{N})/\mathcal{O})} \mathsf{ping}\ l.\mathit{sync}!\langle\rangle \\[2ex] \mid \underbrace{\mathit{sync}?().\ldots.\mathit{sync}?()}_{|\mathbf{loc}(\mathcal{N})|}.x!\langle\rangle \end{array} \right)$$

Then we prove this lemma by contradiction. We analyse all the possible cases why $O \neq \Gamma_O$ and $\mathbf{loc}(\mathcal{N})/O \neq \mathbf{loc}(\mathcal{I}(\Gamma)_{\mathcal{N}})/\mathcal{I}(\Gamma)_O$ and then show that for each of these cases

$$\Gamma_+ \triangleright k_0[\![\mathit{verPubStat}^{\langle \mathcal{N}, O \rangle}(\textsc{succ})]\!] \not\longrightarrow^* \Gamma_+ \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!]$$

□

Proposition 4.2.24 (Definability). Assume that for an arbitrary network representation Γ, the network Γ_+ denotes:

$$\Gamma_+ = \Gamma + k_0 : \mathtt{p}[\mathtt{a}], \textsc{succ} : \mathtt{p}\langle - \rangle, \textsc{fail} : \mathtt{p}\langle - \rangle$$

where k_0, $\textsc{succ}$ and $\textsc{fail}$ are fresh to $\Gamma_{\mathcal{T}}$. Thus, for every external action μ and network representation Γ, every non-empty finite set of names Nm where $\mathbf{names}(\Gamma_{\mathcal{T}}) \subseteq Nm$, every fresh pair of channel names $\textsc{succ}$, $\textsc{fail} \notin Nm$, and every fresh live location name $k_0 \notin Nm$, there exists a system $T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$ with the property that $\Gamma_+ \vdash_{\mathsf{obs}} T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$, such that:

1. $\Gamma \triangleright N \overset{\mu}{\longmapsto} \Gamma' + \mathbf{bn}(\mu) : \tilde{\mathrm{T}} \triangleright N'$ implies
   $\Gamma_+ \triangleright N \,|\, T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0) \longrightarrow^* \Gamma'_+ \triangleright (\nu\, \mathbf{bn}(\mu) : \tilde{\mathrm{T}})\, N' \mid k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu) \rangle]\!]$
2. $\Gamma_+ \triangleright N \,|\, T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0) \longrightarrow^* \Gamma'_+ \triangleright N'$, where $\Gamma'_+ \triangleright N' \Downarrow_{\textsc{succ}@k_0}$, $\Gamma'_+ \triangleright N' \not\Downarrow_{\textsc{fail}@k_0}$ implies that
   $N' \equiv (\nu\, \mathbf{bn}(\mu) : \tilde{\mathrm{T}}) N'' | k_0[\![\textsc{succ}!\langle \mathbf{bn}(\mu) \rangle]\!]$ for some N'' such that $\Gamma \triangleright N \overset{\mu}{\Longmapsto} \Gamma' + \mathbf{bn}(\mu) : \tilde{\mathrm{T}} \triangleright N''$.

Proof. (Outline) We have to prove that the above two clauses are true for all of the three external actions. If μ is $\mathsf{kill} : l$, the test required are:

$$l[\![\mathsf{kill}]\!] \mid k_0[\![\textsc{fail}!\langle\rangle]\!] \mid k_0[\![\mathsf{ping}\; l.\mathsf{ping}\; l.\ulcorner \textsc{fail}?().\textsc{succ}!\langle\rangle \urcorner]\!]$$

If μ is the bound input action $(\tilde{n} : \tilde{\mathrm{S}})l : a?(V)$, the required system is

$$(\nu\, \tilde{n} : \tilde{\mathrm{T}})(l[\![a!\langle V \rangle.\mathsf{go}\; k_0.\textsc{fail}?().\textsc{succ}!\langle\rangle]\!] \,|\, k_0[\![\textsc{fail}!\langle\rangle]\!]) \qquad \text{where } \tilde{\mathrm{S}} = \mathsf{simp}(\tilde{\mathrm{T}})$$

For the output case where μ is $(\tilde{n} : \tilde{\mathrm{S}})l : a!\langle V \rangle$, the required $T^{\mu}(Nm, \textsc{succ}, \textsc{fail}, k_0)$ is

$$k_0[\![\textsc{fail}!\langle\rangle]\!] \mid$$
$$l \left[\!\!\left[a?(X).(\nu\, sync) \left(\begin{array}{l} \prod_{i=1}^{m} \mathsf{if}\; x_i \notin Nm.sync!\langle\rangle \mid \prod_{j=m+1}^{|X|} \mathsf{if}\; x_j = v_j.sync!\langle\rangle \\ \mid \underbrace{sync?()..sync?()}_{|X|}.\mathsf{go}\; k_0.(\nu c) \left(\begin{array}{l} \mathit{verPubStat}^{\mathcal{I} + x_{(m+1)}..x_{|X|} : \mathrm{T}_{(m+1)}..\mathrm{T}_{|X|}}(c) \\ c?().\textsc{fail}?().\textsc{succ}!\langle x_{(m+1)}..x_{|X|} \rangle \end{array} \right) \end{array} \right) \right]\!\!\right]$$

For the sake of presentation ,we once again assume that the first $v_1 \ldots v_m$ in $V = v_1 \ldots v_{|V|}$ in μ are bound, and the remaining $v_{m+1} \ldots v_{|V|}$ are free. □

Theorem 4.2.25 (Soundness and Completeness for typed DπLoc). In typed DπLoc,

- $\mathcal{I} \models \Gamma_1 \triangleright M_1 \approx_{\mathit{ff}} \Gamma_2 \triangleright M_2$ if and only if $\mathcal{I} \models \Gamma_1 \triangleright M_1 \cong_{\mathit{ff}} \Gamma_2 \triangleright M_2$.
- $\mathcal{I} \models \Gamma_1 \triangleright M_1 \approx \Gamma_2 \triangleright M_2$ if and only if $\mathcal{I} \models \Gamma_1 \triangleright M_1 \cong \Gamma_2 \triangleright M_2$. ■

Proof. (Outline) Again, even though it is a laborious process, we here omit the details of the proof, as the result can be derived as a similar case of Theorem 3.5.10. □

Table 22. *β-Transition Rules for Typed DπLoc*

Assuming $\Gamma \vdash l : \textbf{alive}$

(b-in-rep)
$$\frac{}{\Gamma \triangleright l[\![*a?(X).P]\!] \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l[\![a?(X).(P|*a?(Y).P\{^Y/_X\})]\!]}$$

(b-fork)
$$\frac{}{\Gamma \triangleright l[\![P|Q]\!] \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l[\![P]\!] \mid l[\![Q]\!]}$$

(b-eq)
$$\frac{}{\Gamma \triangleright l[\![\text{if } u = u.P\lceil Q \rceil]\!] \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l[\![P]\!]}$$

(b-neq)
$$\frac{}{\Gamma \triangleright l[\![\text{if } u = v.P\lceil Q \rceil]\!] \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l[\![Q]\!]} \; u \neq v$$

(b-new)
$$\frac{}{\Gamma \triangleright l[\![(\nu\, n : \mathtt{T})P]\!] \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright (\nu\, n : \mathtt{T})\, l[\![P]\!]}$$

(b-ngo)
$$\frac{}{\Gamma \triangleright l[\![\text{go } k.P]\!] \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright k[\![\mathbf{0}]\!]} \; \Gamma \nvdash k \leftarrow l$$

(b-nping)
$$\frac{}{\Gamma \triangleright l[\![\text{ping } k.P\lceil Q \rceil]\!] \stackrel{\tau}{\longmapsto}_\beta \Gamma \triangleright l[\![Q]\!]} \; \Gamma \nvdash k \leftarrow l$$

(b-rest)
$$\frac{\Gamma + n : \mathtt{T} \triangleright N \stackrel{\tau}{\longmapsto}_\beta \Gamma' + n : \mathtt{U} \triangleright N'}{\Gamma \triangleright (\nu\, n : \mathtt{T})N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright (\nu\, n : \mathtt{U})N'}$$

(b-par)
$$\frac{\Gamma \triangleright N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright N'}{\begin{array}{c}\Gamma \triangleright N \,|\, M \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright N' \,|\, M \\ \Gamma \triangleright M \,|\, N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright M \,|\, N'\end{array}} \; \Gamma \vdash M$$

4.2.3 Bisimulation up-to proof techniques

In § 4.2.2 we proved that in order to show that two configurations are reduction barb congruent and failure-free reduction barb congruent, it suffices to give a relation that satisfies the requirements of a bisimulation ranging over the respective actions. In this section, we develop auxiliary methods that can relieve some of the burden of exhibiting such relations.

We start by identifying a number of τ actions, which we refer to as β-actions or β-moves, inspired by the work in [JR04, CHR05]. These are denoted as

$$\Gamma \triangleright N \stackrel{\tau}{\longmapsto}_\beta \Gamma' \triangleright N$$

and are defined in Table 22; we also use the notation $\Longmapsto_\beta$ as a shorthand for $\stackrel{\tau}{\longmapsto}{}^*_\beta$. The purpose of these β-moves is to develop up-to bisimulation techniques, by showing that in our witness bisimulations, we can abstract away from matching configurations that denote β-moves. For our specific case, however, the details are slightly more complicated than in [JR04, CHR05] because we deal with distribution and failure. Hence, Table 22 contains local τ-actions that do not involve any interaction with other located processes such as (b-in-rep), (b-fork) and (b-eq), together with τ-actions that involve distribution whose outcome is predetermined to fail because a (permanent) fault has already occurred, such as (b-ngo) and (b-nping).

Table 23. *β-Equivalence Rules for Typed DπLoc*

(b-comm)	$\Gamma \models N\vert M$	$\equiv_f$	$M\vert N$	
(b-assoc)	$\Gamma \models (N\vert M)\vert M'$	$\equiv_f$	$N\vert(M\vert M')$	
(b-unit)	$\Gamma \models N\vert l[\![\mathbf{0}]\!]$	$\equiv_f$	N	
(b-extr)	$\Gamma \models (\nu\, n\!:\!\mathtt{T})(N\vert M)$	$\equiv_f$	$N\vert(\nu\, n\!:\!\mathtt{T})M$	$n \notin \mathbf{fn}(N)$
(b-flip)	$\Gamma \models (\nu\, n\!:\!\mathtt{T})(\nu\, m\!:\!\mathtt{U})N$	$\equiv_f$	$(\nu\, m\!:\!\mathtt{U})(\nu\, n\!:\!\mathtt{T})N$	
(b-inact)	$\Gamma \models (\nu\, n\!:\!\mathtt{T})N$	$\equiv_f$	N	$n \notin \mathbf{fn}(N)$
(b-dead)	$\Gamma \models l[\![P]\!]$	$\equiv_f$	$l[\![Q]\!]$	$\Gamma \nvDash l : \mathbf{alive}$

In addition, to obtain the required results in a distributed setting with failure, we have to define a new structural equivalence, denoted as $\equiv_f$, which takes into consideration the *state of the network* as well. This enables us to obtain confluence for β-moves with respect to actions that change the state of the network and kill locations; we revisit this point in greater detail in the following proofs. The new structural equivalence relation ranges over *configurations* and is the least symmetric contextual relation defined by the rules in Table 23. The new rule worth highlighting is (bs-dead), which allows us to ignore dead code in our structural equivalence relation. Thus for example if for an arbitrary Γ we have $\Gamma \nvDash l : \mathbf{alive}$ then we can conclude $\Gamma \triangleright l[\![P]\!] \,|\, N \equiv_f \Gamma \triangleright l[\![Q]\!] \,|\, N$.

The first property we prove about our β-moves, $\overset{\tau}{\longmapsto}_\beta$, is that they are defined over configurations with the *same* underlying network; this simplifies their handling in subsequent proofs.

Lemma 4.2.26. If $\Gamma \triangleright M \Longmapsto_\beta \Gamma' \triangleright M'$ then $\Gamma = \Gamma'$.

Proof. By simple induction on why $\Gamma \triangleright M \Longmapsto_\beta \Gamma' \triangleright M'$. □

We next show that $\equiv_f$ is a strong bisimulation with respect to the actions defined in § 4.2.2.

Lemma 4.2.27 (β-Structural Equivalence and Strong Bisimulation). If $\Gamma \models M \;\equiv_f\; N$ then

- if $\Gamma \triangleright M \xrightarrow{\mu} \Gamma' \triangleright M'$ then $\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'$ such that $\Gamma \models M' \;\equiv_f\; N'$

Stated otherwise, $\equiv_f \subseteq \sim$

Proof. The proof proceed by induction on why $\Gamma \models M \;\equiv_f\; N$. An outline of a very similar proof was already given for Proposition 2.3.3 □

At this point, we are able to state an important property about β-moves, namely that these internal actions respect the commutative (diamond) property up to β-structural equivalence, as we now state.

Lemma 4.2.28 (Confluence of β-moves). Suppose $\Gamma \triangleright N \overset{\tau}{\longmapsto}_{\beta} \Gamma \triangleright M$. Then for every action $\Gamma \triangleright N \xrightarrow{\mu} \Gamma' \triangleright N'$:

- either $\mu = \tau$ and $\Gamma \triangleright M = \Gamma' \triangleright N'$
- or there is a configuration $\Gamma' \triangleright M'$ such that $\Gamma' \triangleright N' \Longmapsto_{\beta} \equiv_f \Gamma' \triangleright M'$ and $\Gamma \triangleright M \xrightarrow{\mu} \Gamma' \triangleright M'$.

This can be diagrammatically depicted as:

$$\begin{array}{ccc} \Gamma \triangleright N & \overset{\tau}{\longmapsto}_{\beta} & \Gamma \triangleright M \\ \mu \downarrow & & \\ \Gamma' \triangleright N' & & \Gamma' \triangleright M' \end{array} \quad \text{implies} \quad \begin{array}{ccc} \Gamma \triangleright N & \overset{\tau}{\longmapsto}_{\beta} & \Gamma \triangleright M \\ \mu \downarrow & & \downarrow \mu \\ \Gamma' \triangleright N' & \overset{\tau}{\Longmapsto}_{\beta} \equiv_f & \Gamma' \triangleright M' \end{array} \quad \text{or} \quad \mu = \tau \text{ and } \Gamma \triangleright M = \Gamma' \triangleright N'$$

Proof. The proof proceeds by case analysis of the different types of μ and then by induction on the derivation of the β-move.

As an example, we here consider one of the cases, where $\mu = \mathsf{kill} : l$ and a respective key case of the induction on the derivation of the β-move. Thus if

$$\Gamma \triangleright N \overset{\mathsf{kill}:l}{\longmapsto} \Gamma' \triangleright N'$$

then through a lemma such as Lemma 3.5.3, we deduce that $\Gamma \vdash l : \mathbf{alive}$ and $\Gamma \vdash_{\mathsf{obs}} l$, which we abbreviate to

$$\Gamma \vdash_{\mathsf{obs}} l : \mathbf{alive} \tag{4.22}$$

We also deduce that

$$\Gamma' = \Gamma - l \tag{4.23}$$

We now proceed by induction on the length of derivation of

$$\Gamma \triangleright N \overset{\tau}{\longmapsto}_{\beta} \Gamma \triangleright M \tag{4.24}$$

As an example, if the last rule used to derive (4.24) is (b-fork), then we can easily deduce that N and M have the form $l[\![P|Q]\!]$ and $l[\![P]\!]|l[\![Q]\!]$ respectively.

The weak beta move required is the empty move. From (4.22), (4.23) and (l-halt) we deduce

$$\Gamma \triangleright l[\![P]\!]|l[\![Q]\!] \overset{\mathsf{kill}:l}{\longmapsto} \Gamma - l \triangleright l[\![P]\!]|l[\![Q]\!]$$

Finally, it is easy to show that

$$\Gamma - l \triangleright l[\![P|Q]\!] \equiv_f \Gamma - l \triangleright l[\![P]\!]|l[\![Q]\!]$$

using the rule (bs-dead) and the obvious fact that $\Gamma - l \nvdash l : \mathbf{alive}$. □

We also prove that when a configuration performs a β-move, the reduct and redex configurations are weakly bisimilar.

Proposition 4.2.29. Suppose $\Gamma \triangleright N \Longmapsto_{\beta} \Gamma \triangleright M$. Then $\Gamma \models N \approx M$.

Proof. We prove the above statement by defining $\mathcal{R}$ as

$$\mathcal{R} \;=\; \left\{ \;\Gamma \triangleright N \quad, \Gamma \triangleright M \qquad | \; \Gamma \triangleright N \models\!\!\Longrightarrow_\beta \Gamma \triangleright M \;\right\}$$

and showing that $\mathcal{R}$ is a bisimulation. The proof is an easy one and follows as a consequence of Lemma 4.2.28. □

This result provides us with valid equations for reasoning about configurations such as

$$\begin{array}{lll}
\Gamma \models (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, l[\![P \,|\, Q]\!] & \approx\; (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, l[\![P]\!] \,|\, l[\![Q]\!] & \\
\Gamma \models (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, l[\![\mathsf{if}\; v = v.P\lceil Q\rceil]\!] & \approx\; (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, l[\![P]\!] & \\
\Gamma \models (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, l[\![(\nu\,n:\mathrm{T})P]\!] & \approx\; (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, (\nu\,n:\mathrm{T})l[\![P]\!] & \\
\Gamma \models (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, l[\![\mathsf{go}\; k.P]\!] & \approx\; (\nu\,\tilde{n}:\tilde{\mathrm{T}})\,N \,|\, k[\![\mathbf{0}]\!] & \text{if } \Gamma + \tilde{n}:\tilde{\mathrm{T}} \nvdash k : \textbf{alive}
\end{array}$$

But more importantly, it also provides us with a powerful method for approximating bisimulations. From here onwards, we focus our proofs for the most involving bisimulation $\approx$, in the knowledge that some of these results, such as Posposition 4.2.29, can be immediately extrapolated to $\approx_{\mathit{ff}}$ using Proposition 4.2.19, while others can be proved for $\approx_{\mathit{ff}}$ using the same method of the proofs for $\approx$.

Definition 4.2.30 (Bisimulation up-to β-moves). Bisimulation up-to β-moves, denoted as $\approx_\beta$, is the largest typed relation between configurations such that whenever we have $\Gamma_1 \triangleright M_1 \approx_\beta \Gamma_2 \triangleright M_2$ then

- $\Gamma_1 \triangleright M_1 \xrightarrow{\mu} \Gamma_1' \triangleright M_1'$ implies $\Gamma_2 \triangleright M_2 \stackrel{\hat{\mu}}{\Longrightarrow} \Gamma_2' \triangleright M_2'$ such that $\Gamma_1' \triangleright M_1' \,\mathcal{A}_l \circ \approx_\beta \circ \approx \Gamma_2' \triangleright M_2'$
- $\Gamma_2 \triangleright M_2 \xrightarrow{\mu} \Gamma_2' \triangleright M_2'$ implies $\Gamma_1 \triangleright M_1 \stackrel{\hat{\mu}}{\Longrightarrow} \Gamma_1' \triangleright M_1'$ such that $\Gamma_2' \triangleright M_2' \,\mathcal{A}_l \circ \approx_\beta \circ \approx \Gamma_1' \triangleright M_1'$

where $\mathcal{A}_l$ is the relation $\models\!\!\Longrightarrow_\beta \circ \equiv_f$.

In the approximate bisimulation $\approx_\beta$ just defined, one can match an action $\Gamma_1 \triangleright M_1 \xrightarrow{\mu} \Gamma_1' \triangleright M_1'$ by finding a β-derivative of $\Gamma_1' \triangleright M_1'$, that is $\Gamma_1' \triangleright M_1' \models\!\!\Longrightarrow_\beta \Gamma_1' \triangleright M_1''$, and a weak matching action $\Gamma_2 \triangleright M_2 \stackrel{\hat{\mu}}{\Longrightarrow} \Gamma_2' \triangleright M_2'$ such that, up to structural equivalence on the one side and up-to bisimilarity on the other, the pairs $\Gamma_1' \triangleright M_1''$ and $\Gamma_2' \triangleright M_2'$ are once more related. Intuitively then, in any such relation satisfying $\approx_\beta$, a configuration can represent all the configurations to which it can evolve using β-moves. We justify the use of $\approx_\beta$ by proving Proposition 4.2.32; this in turn uses the following lemma, which we extract from the proof.

Lemma 4.2.31. Suppose $\Gamma_1 \triangleright M_1 \approx_\beta \Gamma_2 \triangleright M_2$ and $\Gamma_1 \triangleright M_1 \stackrel{\widehat{\mu}}{\models\!\!\Longrightarrow} \Gamma_1' \triangleright M_1'$. Then $\Gamma_2 \triangleright M_2 \stackrel{\widehat{\mu}}{\models\!\!\Longrightarrow} \Gamma_2' \triangleright M_2'$, such that $\Gamma_1' \triangleright M_1' \approx \circ \approx_\beta \circ \approx \Gamma_2' \triangleright M_2'$

Proof. We proceed by induction on the length of $\Gamma_1 \triangleright M_1 \stackrel{\widehat{\mu}}{\models\!\!\Longrightarrow} \Gamma_1' \triangleright M_1'$:

- The base case, when the length is zero and $\Gamma_1 \triangleright M_1 = \Gamma_1' \triangleright M_1'$ is trivial.

- There are two inductive cases. We here focus on one case where

$$\Gamma_1 \triangleright M_1 \xmapsto{\tau} \Gamma_1^1 \triangleright M_1^1 \stackrel{\widehat{\mu}}{\Longmapsto} \Gamma_1' \triangleright M_1' \tag{4.25}$$

and leave the other (similar) case for the interested reader.
By Definition 4.2.30 (for $\approx_\beta$),

$$\exists\, \Gamma_2 \triangleright M_2 \Longmapsto \Gamma_2^1 \triangleright M_2^1 \text{ such that } \Gamma_1^1 \triangleright M_1^1 \; \mathcal{A}_l \circ \approx_\beta \circ \approx \Gamma_2^1 \triangleright M_2^1 \tag{4.26}$$

By expanding (4.25) and (4.26) we have the following diagram to complete

$$\begin{array}{ccccccccc}
\Gamma_1^1 \triangleright M_1^1 & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1^1 \triangleright M_1^2 & \equiv_f & \Gamma_1^1 \triangleright M_1^3 & \approx_\beta & \Gamma_2^2 \triangleright M_2^2 & \approx & \Gamma_2^1 \triangleright M_2^1 \\
\widehat{\mu}\Big\Downarrow & & & & & & & & \\
\Gamma_1' \triangleright M_1' & & & & & & & &
\end{array}$$

Through Lemma 4.2.28 (confluence of β-moves) we can fill the first part of the diagram as

$$\begin{array}{ccccccccc}
\Gamma_1^1 \triangleright M_1^1 & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1^1 \triangleright M_1^2 & \equiv_f & \Gamma_1^1 \triangleright M_1^3 & \approx_\beta & \Gamma_2^2 \triangleright M_2^2 & \approx & \Gamma_2^1 \triangleright M_2^1 \\
\widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & & & & & \\
\Gamma_1' \triangleright M_1' & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1' \triangleright M_1^4 & & & & & &
\end{array}$$

and by Lemma 4.2.27 we can fill the third part

$$\begin{array}{ccccccccc}
\Gamma_1^1 \triangleright M_1^1 & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1^1 \triangleright M_1^2 & \equiv_f & \Gamma_1^1 \triangleright M_1^3 & \approx_\beta & \Gamma_2^2 \triangleright M_2^2 & \approx & \Gamma_2^1 \triangleright M_2^1 \\
\widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & & & \\
\Gamma_1' \triangleright M_1' & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1' \triangleright M_1^4 & \equiv_f & \Gamma_1' \triangleright M_1^5 & & & &
\end{array}$$

By induction we fill in the fourth part

$$\begin{array}{ccccccccc}
\Gamma_1^1 \triangleright M_1^1 & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1^1 \triangleright M_1^2 & \equiv_f & \Gamma_1^1 \triangleright M_1^3 & \approx_\beta & \Gamma_2^2 \triangleright M_2^2 & \approx & \Gamma_2^1 \triangleright M_2^1 \\
\widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & \\
\Gamma_1' \triangleright M_1' & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1' \triangleright M_1^4 & \equiv_f & \Gamma_1' \triangleright M_1^5 & \approx \circ \approx_\beta \circ \approx & \Gamma_2^3 \triangleright M_2^3 & &
\end{array}$$

And finally we complete the diagram by the definition of $\approx$

$$\begin{array}{ccccccccc}
\Gamma_1^1 \triangleright M_1^1 & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1^1 \triangleright M_1^2 & \equiv_f & \Gamma_1^1 \triangleright M_1^3 & \approx_\beta & \Gamma_2^2 \triangleright M_2^2 & \approx & \Gamma_2^1 \triangleright M_2^1 \\
\widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow & & \widehat{\mu}\Big\Downarrow \\
\Gamma_1' \triangleright M_1' & \xmapsto{\tau}{}^{*}_{\beta} & \Gamma_1' \triangleright M_1^4 & \equiv_f & \Gamma_1' \triangleright M_1^5 & \approx \circ \approx_\beta \circ \approx & \Gamma_2^3 \triangleright M_2^3 & \equiv_f & \Gamma_2' \triangleright M_2'
\end{array}$$

The required result follows from the above completed diagram and the fact that $\Longmapsto_\beta \subseteq \approx$, (Proposition 4.2.29), and $\equiv_f \subseteq \approx$ (Lemma 4.2.27).

□

Proposition 4.2.32 (Inclusion of bisimulation up-to β-moves). If $\Gamma_1 \triangleright M_1 \approx_\beta \Gamma_2 \triangleright M_2$ then $\Gamma_1 \triangleright M_1 \approx \Gamma_2 \triangleright M_2$

Proof. We prove the above proposition by defining the relation $\mathcal{R}$ as

$$\mathcal{R} \;=\; \left\{ \Gamma_1 \triangleright M_1 \quad, \Gamma_2 \triangleright M_2 \quad \middle|\, \Gamma_1 \triangleright M_1 \approx \circ \approx_\beta \circ \approx \Gamma_2 \triangleright M_2 \right\}$$

and show that $\mathcal{R} \subseteq \approx$. The required result can then be extracted from this result by considering the special cases where the $\approx$ on either side are the identity relations.

Assume $\Gamma_1 \triangleright M_1 \xrightarrow{\mu} \Gamma_1' \triangleright M_1'$.

By our definition of $\mathcal{R}$, $\exists M_1''$ and $\Gamma_2' \triangleright M_2'$ such that

$$\Gamma_1 \triangleright M_1 \approx \Gamma_1'' \triangleright M_1'' \tag{4.27}$$

$$\Gamma_1'' \triangleright M_1'' \approx_\beta \Gamma_2' \triangleright M_2' \tag{4.28}$$

$$\Gamma_2' \triangleright M_2' \approx \Gamma_2 \triangleright M_2 \tag{4.29}$$

From (4.27) and the definition of bisimulation we know

$$\Gamma_1'' \triangleright M_1'' \stackrel{\widehat{\mu}}{\Longrightarrow} \Gamma_1''' \triangleright M_1''' \quad \text{such that } \Gamma_1' \triangleright M_1' \approx \Gamma_1''' \triangleright M_1''' \tag{4.30}$$

By Lemma 4.2.31, (4.30), and (4.28) we know

$$\Gamma_2' \triangleright M_2' \stackrel{\widehat{\mu}}{\Longrightarrow} \Gamma_2'' \triangleright M_2'' \;\; \text{such that } \Gamma_1''' \triangleright M_1''' \approx \circ \approx_\beta \circ \approx \Gamma_2'' \triangleright M_2'' \tag{4.31}$$

and by induction on the number of actions in (4.31) and (4.29) we also conclude

$$\Gamma_2 \triangleright M_2 \stackrel{\widehat{\mu}}{\Longrightarrow} \Gamma_2''' \triangleright M_2''' \;\; \text{such that } \Gamma_2'' \triangleright M_2'' \approx \Gamma_2''' \triangleright M_2''' \tag{4.32}$$

which is our matching move.

Finally $\Gamma_1' \triangleright M_1'$ and $\Gamma_2''' \triangleright M_2'''$ are related in $\mathcal{R}$ by (4.30), (4.31), (4.32) and the transitivity of $\approx$. □

In the sequel, we will always give bisimulations up-to β-moves for both $\approx$ and $\approx_{ff}$; we refer to the end of § 4.3 for such examples.

4.3 Defining Fault Tolerance

We now turn our attention to the definition of fault tolerance for typed DπLoc terms. The server implementations server_2 and server_3 discussed in Example 4.1.1 are our sole motivating examples of fault tolerant systems so far; they attain fault tolerance by masking faults through *active* replication while at the same time, avoiding the use of any fault detection mechanisms. Before we embark on the actual definition of fault tolerance, we overview some examples that use *passive* replication and *fault detection*.

Example 4.3.1. [Passive replication with simple failure detection] The server implementation srvPng_2 defined below, uses two identical replicas of the distributed database at

k_1 and k_2, similar to server_2; the way it manages these replicas is however different from server_2.

$$\mathsf{srvPng}_2 \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[\begin{array}{r} serv?(x,y).\mathsf{ping}\, k_1.\mathsf{go}\, k_1.data!\langle x,y,l\rangle \\ \lceil \mathsf{go}\, k_2.data!\langle x,y,l\rangle\rceil \end{array} \right]\!\!\right] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\, z\,.y!\langle f(x)\rangle]\!] \\ \mid k_2[\![data?(x,y,z).\mathsf{go}\, z\,.y!\langle f(x)\rangle]\!] \end{array} \right)$$

More specifically, srvPng_2 treats the replica at k_1 as *primary* replica and the one at k_2 as a *secondary* (backup) replica. The coordinating proxy at l uses the ping construct as a *simple failure detector* to check the state of the primary replica: if k_1 is alive, the request is sent to the replica at k_1 and the secondary replica at k_2 is *not invoked*; if, on the other hand, the primary replica is dead, then the passive replica at k_2 is promoted to a primary replica and the request is sent to it. This implementation saves on *time redundancy* since for any request, only one replica is invoked. ■

It turns out that srvPng_2 is less reliable than server_2, in spite of the fact that it also uses two replicas. In fact, srvPng_2 might still block even if only one fault occurs during computation. More formally,

$$\exists\, \Gamma', N \text{ such that } \Gamma \triangleright \mathsf{srvPng}_2 \,|\, \mathsf{client} \,|\, k_1[\![\mathsf{kill}]\!] \Longrightarrow \Gamma' \triangleright N \text{ where } \Gamma' \triangleright N \not\Downarrow_{ret@l}$$

This happens when srvPng_2 receives a request from client, tests for the liveness of the primary replica, determines that it is still alive and then sends the request to it. If the fault on k_1 is induced after the ping test, but before the replica at k_1 sends back the answer to the proxy at l, srvPng_2 blocks. We have already seen that this never happens in the case of server_2, irrespective of when the fault is induced. This is not to say that srvPng_2 does not exhibit any fault tolerant behaviour. In fact, if we consider a milder form of *static* fault induction, it exhibits the same fault tolerant behaviour as server_2. More formally,

$$\text{for } i = 1..3,\ \forall\, \Gamma', N \text{ such that } (\Gamma - k_i) \triangleright \mathsf{srvPng}_2 \,|\, \mathsf{client} \Longrightarrow \Gamma' \triangleright N \text{ then } \Gamma' \triangleright N \Downarrow_{ret@l}$$
and
$$\text{for } i = 1..3,\ \forall\, \Gamma', N \text{ such that } (\Gamma - k_i) \triangleright \mathsf{server}_2 \,|\, \mathsf{client} \Longrightarrow \Gamma' \triangleright N \text{ then } \Gamma' \triangleright N \Downarrow_{ret@l}$$

Example 4.3.2. The $\mathsf{monitor}$ construct, defined earlier in Example 3.2.2, fits adequately into the well established theory of distributed systems and maps closely to the notion of failure detectors introduced in [CT96]; it turns out that $\mathsf{monitor}$ satisfies the characteristics of a *perfect failure detector* ($\mathcal{P}$), which always gives a correct view of the state of the network. Using the new monitor construct, we modify $\mathsf{srvPing}_2$, swapping the simple failure detection using ping with $\mathsf{monitor}$

$$\mathsf{srvPFD}_2 \Leftarrow (\nu\, data)\left(\begin{array}{l} l\left[\!\!\left[serv?(x,y).(\nu sync)\left(\begin{array}{l} \mathsf{go}\, k_1.data!\langle x, sync, l\rangle \\ \mid \mathsf{monitor}\, k_1\lceil \mathsf{go}\, k_2.data!\langle x, sync, l\rangle\rceil \\ \mid sync?(z).y!\langle z\rangle \end{array} \right) \right]\!\!\right] \\ \mid k_1[\![data?(x,y,z).\mathsf{go}\, z\,.y!\langle f(x)\rangle]\!] \\ \mid k_2[\![data?(x,y,z).\mathsf{go}\, z\,.y!\langle f(x)\rangle]\!] \end{array} \right)$$

The new server implementation, srvPFD_2, still uses *passive* replication in the sense that the replica at k_1 is still the *primary* replica and is queried first. The server proxy, however, launches a parallel monitor on the primary replica, and the *repreated tests* allows it to recover and promote the secondary replica at k_2 to a primary one, when (and if) the primary replica fails. In order to keep our implementation simple, we do not alter the internal workings of monitor to synchronise it with the reply from the replica at k_1. As a result, we might have the sequence of events where the primary replica returns a result at l, then k_1 becomes dead and then the monitor launches a request to the secondary replica at k_2. In this case, we run the risk of receiving two replies at l, which would be detectable by the observer. We thus use the same synchrnonisation mechanism for multiple replies using the scoped channel *sync*, already used in server_2 and server_3. ■

It turns out that srvPFD_2 is as fault tolerant as server_2 is, in the sense that no observer, limited to location l, can tell the two servers apart if at most one fault occurs at either k_1, k_2 or k_3. More specifically,

$$\text{for } i = 1..3,\ \forall\, \Gamma', N \text{ such that } \Gamma \triangleright \mathsf{srvPFD}_2 \,|\, \mathsf{client} \,|\, k_i[\![\mathsf{kill}]\!] \Longrightarrow \Gamma' \triangleright N \text{ then } \Gamma' \triangleright N \Downarrow_{ret@l}$$

At this point, we finally turn our attention to the actual definitions of fault tolerance, where we formalise all the intuitions given so for Examples 4.1.1, 4.3.1 and 4.3.2; we also use these examples as testbeds for our definitions. The fault tolerance definitions we give are based on two concepts: *controlled fault injection on confined locations* hosting redundant computation and the respective *difference in behaviour observed at public locations* as a result of the fault injections. We thus start to build this definition incrementally, by defining alternative methods for fault injection.

Definition 4.3.3 (Fault Contexts). We identify two ways how to inject faults (thus failures) on a typed network representation Γ: either *statically*, using the operation $\Gamma - l$ or else *dynamically*, using the located process $l[\![\mathsf{kill}]\!]$. The difference between these two methods is that the former, inject the fault immediately, whereas the latter can induce the fault asynchronously, at any stage during reduction.

Based on these two methods of fault injection, we define what we call *fault contexts*, instantiated with a parameter n, denoting the *maximum* number of faults that the context can induce. Thus:

- A *Static* n-fault context, or n-s.f.context for short, is denoted as an operation on typed network representation $F^n_S(x)$ and is defined as

$$F^n_S(x) \;=\; x - l_1 \ldots - l_n \;\text{ or } x - l_{1..n} \text{ for short}$$

- A *Dynamic* n-fault context, or n-d.f. context for short, is a system denoted as F^n_D and defined as

$$F^n_D \;=\; l_1[\![\mathsf{kill}]\!] \,|\, \ldots \,|\, l_n[\![\mathsf{kill}]\!] \;\text{ or } \prod_{i=1}^{n} l_i[\![\mathsf{kill}]\!] \text{ for short}$$

We here note that all the locations in a fault context need not be distinct, meaning that an n-fault context kills *at most* n locations. If there are duplicate locations in a fault context ($l_i = l_j$ for some $i, j \in 1..n$ where $i \neq j$), they effectively amount to a *single* fault, since a location cannot be killed more than once. As a shorthand, we often also remove the differentiating marks S and D denoting static and dynamic, and denote a fault context as F^n, where these can be inferred from the text.

Finally, we also define the notion of a *valid* fault context with respect to a typed network representation Γ, where all the faults are injected on *confined* locations: this denoted as $\Gamma \vdash F^n$ and defined as

$$\Gamma \vdash F^n \stackrel{\text{def}}{=} \forall\, l \in \mathbf{fn}(F^n) \text{ then } \Gamma \vdash l : \mathsf{c}$$

■

As we shall see in the sequel, the way we induce failure influences the kind of fault tolerance we define. In particular, we now define two forms of fault tolerance called *static fault tolerance* and *dynamic fault tolerance*; as the name implies, the former injects faults using static fault contexts whereas the latter uses dynamic fault contexts. These definitions can also be viewed as defining *perfect* fault tolerance, by which we mean that we require that *all* the observable barbs of a configuration are preserved for *any* reduction sequence when faults are injected. For this reason, it seems appropriate to reuse the equivalence relation $\cong_{ff}$ defined earlier in § 4.2.14; this allows us to keep the state of the public locations stable while we alter the state of the confined locations internally.

Definition 4.3.4 (n-Static Fault Tolerance). A (well-formed) configuration $\Gamma \triangleright N$ is said to be n-static fault tolerant, or n-s.f.t., if and only iff

$$\forall\, F^n_S(x) \text{ such that } \Gamma \vdash F^n_S(x) \text{ we have } \;\Gamma \triangleright N \;\cong_{ff}\; (F^n_S(\Gamma)) \triangleright N$$

■

Example 4.3.5. According to Definition 4.3.4, for

$$\Gamma \;=\; \langle\, \{\, l{:}\mathtt{p},\;\; k_{1..3}{:}\mathtt{c},\;\; req{:}\mathtt{p}\langle \mathtt{p}, \mathtt{p}\langle - \rangle\rangle,\;\; ret{:}\mathtt{p}\langle - \rangle \,\}, \emptyset \,\rangle$$

we can conclude that the configurations $\Gamma \triangleright \mathsf{server}_2$, $\Gamma \triangleright \mathsf{server}_3$, $\Gamma \triangleright \mathsf{srvPng}_2$ and $\Gamma \triangleright \mathsf{srvPFD}_2$ are all 1-static fault tolerant, whereas $\Gamma \triangleright \mathsf{server}_1$ is not, because

$$\Gamma \triangleright \mathsf{server}_1 \;\not\cong_{ff}\; (\Gamma - k_1) \triangleright \mathsf{server}_1$$

Moreover, we can also conclude that $\Gamma \triangleright \mathsf{server}_3$ is also 2-static fault tolerant, whereas all the rest are not; for instance,

$$\Gamma \triangleright \mathsf{server}_2 \;\not\cong_{ff}\; (\Gamma - k_1, k_2) \triangleright \mathsf{server}_2$$

We can verify that a configuration is n-static fault tolerant by using the bisimulations developed in § 4.2.2: thus, to show that $\Gamma \triangleright \mathsf{srvPng}_2$ is 1-static fault tolerant, we provide witness relations satisfying

$$\Gamma \triangleright \mathsf{srvPng}_2 \;\approx_{ff}\; \Gamma - k_1 \triangleright \mathsf{srvPng}_2 \tag{4.33}$$

$$\Gamma \triangleright \mathsf{srvPng}_2 \;\approx_{ff}\; \Gamma - k_2 \triangleright \mathsf{srvPng}_2 \tag{4.34}$$

$$\Gamma \triangleright \mathsf{srvPng}_2 \;\approx_{ff}\; \Gamma - k_3 \triangleright \mathsf{srvPng}_2 \tag{4.35}$$

We here give the witness relation for (4.33) and leave the other simpler relations for the interested reader: Thus, the witness relation is $\mathcal{R}$ defined as

$$\mathcal{R} \stackrel{\text{def}}{=} \{\langle \Gamma \triangleright \mathsf{srvPng}_2, \Gamma - k_1 \triangleright \mathsf{srvPng}_2 \rangle\} \cup \left(\bigcup_{u,v \in \text{NAMES}} \mathcal{R}'(u,v) \right)$$

$$\mathcal{R}'(u,v) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \Gamma \triangleright (\nu d) l[\![\mathsf{Png}(u,v)]\!] \mid \mathsf{R}_1 \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) l[\![\mathsf{Png}(u,v)]\!] \mid \mathsf{R}_1 \mid \mathsf{R}_2 \\ \Gamma \triangleright (\nu d) l[\![\mathsf{Q}_1(u,v)]\!] \mid \mathsf{R}_1 \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) l[\![\mathsf{Q}_2(u,v)]\!] \mid \mathsf{R}_1 \mid \mathsf{R}_2 \\ \Gamma \triangleright (\nu d) k_1[\![d!\langle u,v,l\rangle]\!] \mid \mathsf{R}_1 \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) k_2[\![d!\langle u,v,l\rangle]\!] \mid \mathsf{R}_1 \mid \mathsf{R}_2 \\ \Gamma \triangleright (\nu d) k_1[\![\mathsf{go}\ l.v!\langle f(u)\rangle]\!] \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) \mathsf{R}_1 \mid k_2[\![\mathsf{go}\ l.v!\langle f(u)\rangle]\!] \\ \Gamma \triangleright (\nu d) l[\![v!\langle f(u)\rangle]\!] \mid \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) \mathsf{R}_1 \mid l[\![v!\langle f(u)\rangle]\!] \\ \Gamma \triangleright (\nu d) \mathsf{R}_2 & , \Gamma - k_1 \triangleright (\nu d) \mathsf{R}_1 \end{array} \right\}$$

where d stands for *data* and

$$\begin{array}{rcl} \mathsf{Png}(x,y) & \Leftarrow & \mathsf{ping}\ k_1.\mathsf{Q}_1(x,y)\lceil \mathsf{Q}_2(x,y)\rceil \\ \mathsf{Q}_i(x,y) & \Leftarrow & \mathsf{go}\ k_i.d!\langle x,y,l\rangle \\ \mathsf{R}_i & \Leftarrow & k_i[\![d?(x,y,z).\mathsf{go}\ z\ .y!\langle f(x)\rangle]\!] \end{array}$$

$\mathcal{R}$ is the union of all the relations $\mathcal{R}'(x,y)$ where the variables x, y are parameterised by names $u, v \in$ NAMES. In turn, $\mathcal{R}'(x,y)$ is an up-to-β-moves failure free bisimulation relating $\Gamma \triangleright \mathsf{srvPng}_2$ and $\Gamma - k_1 \triangleright \mathsf{srvPng}_2$. In this mapping one can see that while all the requests are serviced by the primary replica at k_1 in $\Gamma \triangleright \mathsf{srvPng}_2$, they are serviced by the secondary replica at k_2 in $\Gamma - k_1 \triangleright \mathsf{srvPng}_2$. ■

Definition 4.3.6 (n-Dynamic Fault Tolerance). A (well-formed) configuration $\Gamma \triangleright N$ is said to be n-dynamic fault tolerant, or n-d.f.t., if and only if

$$\forall\ F^n_D \text{ such that } \Gamma \vdash F^n_D \text{ we have } \Gamma \triangleright N \cong_{ff} \Gamma \triangleright N \mid F^n_D$$

■

Example 4.3.7. According to Definition 4.3.6, for

$$\Gamma = \langle \{l\!:\!\mathtt{p},\ k_{1..3}\!:\!\mathtt{p},\ req\!:\!\mathtt{p}\langle \mathtt{p}, \mathtt{p}\langle - \rangle\rangle,\ ret\!:\!\mathtt{p}\langle - \rangle\},\ \emptyset \rangle$$

we can conclude that the configurations $\Gamma \triangleright \mathsf{server}_2$, $\Gamma \triangleright \mathsf{server}_3$ and $\Gamma \triangleright \mathsf{srvPFD}_2$ are all 1-dynamic fault tolerant, whereas $\Gamma \triangleright \mathsf{server}_1$ and $\Gamma \triangleright \mathsf{srvPng}_2$ are not. For instance we can show that

$$\Gamma \models \mathsf{srvPng}_2 \not\cong_{ff} \mathsf{srvPng}_2 \mid k_1[\![\mathsf{kill}]\!]$$

We can also conclude that $\Gamma \triangleright \mathsf{server}_3$ is also 2-dynamic fault tolerant, whereas all the rest are not; for instance,

$$\Gamma \triangleright \mathsf{server}_2 \not\cong_{ff} \Gamma \triangleright \mathsf{server}_2 \mid k_1[\![\mathsf{kill}]\!] \mid k_2[\![\mathsf{kill}]\!]$$

Once again, we can use the bisimulations developed in § 4.2.2 to verify that a configuration is n-dynamic fault tolerant: thus, to show that $\Gamma \triangleright \mathsf{server}_2$ is 1-dynamic fault tolerant, we

provide witness relations satisfying

$$\Gamma \triangleright \mathsf{server}_2 \approx_{ff} \Gamma \triangleright \mathsf{server}_2 \,|\, k_1[\![\mathsf{kill}]\!] \tag{4.36}$$

$$\Gamma \triangleright \mathsf{server}_2 \approx_{ff} \Gamma \triangleright \mathsf{server}_2 \,|\, k_2[\![\mathsf{kill}]\!] \tag{4.37}$$

$$\Gamma \triangleright \mathsf{server}_2 \approx_{ff} \Gamma \triangleright \mathsf{server}_2 \,|\, k_3[\![\mathsf{kill}]\!] \tag{4.38}$$

Below we give the witness relation $\mathcal{R}$ for (4.36); we leave the other simpler relations for the interested reader. It is the union of all the relations $\mathcal{R}'(x,y)$ where the variables x, y are parameterised by names $u, v \in$ NAMES.

$$\mathcal{R} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \langle \Gamma \triangleright \mathsf{server}_2,\, \Gamma \triangleright \mathsf{server}_2 \,|\, k_1[\![\mathsf{kill}]\!] \rangle, \\ \langle \Gamma \triangleright \mathsf{server}_2,\, \Gamma - k_1 \triangleright \mathsf{server}_2 \rangle \end{array} \right\} \cup \left(\bigcup_{u,v \in \textsc{Names}} \mathcal{R}'(u,v) \right)$$

The relation $\mathcal{R}'(x,y)$ is an up-to-β-moves failure free bisimulation relating $\Gamma \triangleright \mathsf{server}_2$ and $\Gamma \triangleright \mathsf{server}_2 \,|\, k_1[\![\mathsf{kill}]\!]$. The mapping of the intermediary states in this relation is based on the separation of the sub-systems making up server_2 and its derivatives into two classes:

- sub-systems whose behaviour is independent of the state of k_i for $i = 1..2$, that is the locations that may potentially fail. Such an example is the located process $l[\![\mathsf{sync}?(x).y!\langle x \rangle]\!]$, denoted as $l[\![\mathsf{S}(y)]\!]$ below.
- sub-systems whose behaviour depends directly on the state of k_i for $i = 1..2$: typically theses sub-systems are
  - located processes that intend to *go to* k_i, such as the queries sent to the database replica $\mathsf{go}\, k_i.d!\langle x, y, l \rangle$, denoted as $\mathsf{Q}_i(x,y)$ below.
  - processes that *reside at* k_i, such as the database replicas themselves, denoted as R_i below.
  - located processes that have *migrated from* k_i, such as replies from these replicas, $l[\![y!\langle f(x) \rangle]\!]$.

To relate the sub-systems of the second kind (those dependent on the state of k_i), $\mathcal{R}'(x,y)$ uses three relations ranging over systems, namely

- $\mathcal{R}_i^{Id}(x,y)$, an identity relation.
- $\mathcal{R}_i^{\mathbf{0}}(x,y)$ mapping all the left-hand located processes depending on k_i to the null process at k_i on the right hand side, $k_i[\![\mathbf{0}]\!]$, typically used when k_i is killed.
- $\mathcal{R}_i^{\geq}(x,y)$ mapping all the left-hand located processes depending on k_i to the null process at l on the right hand side, $l[\![\mathbf{0}]\!]$, denoting the right hand side state where the replica at k_i has been successfully queried, the answer has been successfully returned and consumed by $l[\![\mathsf{S}(y)]\!]$.

Sub-systems of the second kind are related with these three relations in $\mathcal{R}'(x,y)$ depending on two factors, that is the state of k_1 and whether the global system is in a position to output an answer back to the observer.

- As long as k_1 is alive, then relate the sub-systems depending on k_1 and those depending on k_2 using $\mathcal{R}_i^{Id}(x,y)$ for $i = 1..2$.

- When k_1 dies, then we refer to the second criteria. If the global system is *not* in a position to output an answer back to the observer, derived from the fact that $l[\![\mathsf{S}(y)]\!]$ has not yet reduced, then we relate the sub-systems depending on k_1 using $\mathcal{R}^{\mathbf{0}}_1(x,y)$, exploiting β-moves such as those using (b-ngo) and structural rules such as (bs-dead) from Table 22 and Table 23 respectively; the sub-systems depending on k_2 are still related using $\mathcal{R}^{Id}_i(x,y)$.

- If the global system is in a position to output an answer back to the observer ($l[\![y!\langle f(x)\rangle]\!]$), or has already done so, then we then we relate the sub-systems depending on k_1 using $\mathcal{R}^{\mathbf{0}}_1(x,y)$. We also map the sub-systems depending on k_2 using $\mathcal{R}^{\geq}_2(x,y)$, based on the idea that if k_1 died before servicing the query, then the only way we can output an answer back to the observer is through the complete servicing of the replica at k_2.

There is one exception to the cases when k_1 is dead and the global system is in a position to output an answer back to the observer; this is when the answer returned by the global system on the right is from the replica at k_1. In this case, we simply match it by the identity relation for both sub-systems depending on k_1 and k_2.

$$
\mathcal{R}'(x,y) \stackrel{\text{def}}{=} \left\{
\begin{array}{lll}
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l[\![\mathsf{S}(y)]\!] \\ |\,M_1\,|\,M_2 \end{pmatrix} & ,\Gamma \triangleright \begin{array}{l} (\nu d,s)\begin{pmatrix} l[\![\mathsf{S}(y)]\!] \\ |\,N_1\,|\,N_2 \end{pmatrix} \\ |\,k_1[\![\mathsf{kill}]\!] \end{array} & \left|\begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array}\right. \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l[\![\mathsf{S}(y)]\!] \\ |\,M_1\,|\,M_2 \end{pmatrix} & ,\Gamma{-}k_1 \triangleright (\nu d,s)\begin{pmatrix} l[\![\mathsf{S}(y)]\!] \\ |\,N_1\,|\,N_2 \end{pmatrix} & \left|\begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\mathbf{0}}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array}\right. \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ |\,M_1\,|\,M_2 \end{pmatrix} & ,\Gamma \triangleright \begin{array}{l} (\nu d,s)\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ |\,N_1\,|\,N_2 \end{pmatrix} \\ |\,k_1[\![\mathsf{kill}]\!] \end{array} & \left|\begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array}\right. \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ |\,M_1\,|\,M_2 \end{pmatrix} & ,\Gamma{-}k_1 \triangleright (\nu d,s)\begin{pmatrix} l[\![y!\langle f(x)\rangle]\!] \\ |\,N_1\,|\,N_2 \end{pmatrix} & \left|\begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\mathbf{0}}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{\geq}_2(x,s) \end{array}\right. \\
\Gamma \triangleright (\nu d,s)\big(\, M_1\,|\,M_2 \,\big) & ,\Gamma \triangleright \begin{array}{l} (\nu d,s)\big(\, N_1\,|\,N_2 \,\big) \\ |\,k_1[\![\mathsf{kill}]\!] \end{array} & \left|\begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array}\right. \\
\Gamma \triangleright (\nu d,s)\big(\, M_1\,|\,M_2 \,\big) & ,\Gamma{-}k_1 \triangleright (\nu d,s)\big(\, N_1\,|\,N_2 \,\big) & \left|\begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\mathbf{0}}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{\geq}_2(x,s) \end{array}\right.
\end{array}
\right\}
$$

$$
\mathcal{R}^{\mathbf{0}}_i(x,y) \stackrel{\text{def}}{=} \left\{
\begin{array}{ll}
\mathsf{Q}_i(x,y)\,|\,\mathsf{R}_i & ,k_i[\![\mathbf{0}]\!] \\
k_i[\![d!\langle x,y,l\rangle]\!]\,|\,\mathsf{R}_i & ,k_i[\![\mathbf{0}]\!] \\
k_i[\![\mathsf{go}\; l.y!\langle f(x)\rangle]\!] & ,k_i[\![\mathbf{0}]\!] \\
l[\![y!\langle f(x)\rangle]\!] & ,k_i[\![\mathbf{0}]\!] \\
l[\![\mathbf{0}]\!] & ,k_i[\![\mathbf{0}]\!]
\end{array}
\right\}
\qquad
\mathcal{R}^{\geq}_i(x,y) \stackrel{\text{def}}{=} \left\{
\begin{array}{ll}
\mathsf{Q}_i(x,y)\,|\,\mathsf{R}_i & ,l[\![\mathbf{0}]\!] \\
k_i[\![d!\langle x,y,l\rangle]\!]\,|\,\mathsf{R}_i & ,l[\![\mathbf{0}]\!] \\
k_i[\![\mathsf{go}\; l.y!\langle f(x)\rangle]\!] & ,l[\![\mathbf{0}]\!] \\
l[\![y!\langle f(x)\rangle]\!] & ,l[\![\mathbf{0}]\!] \\
l[\![\mathbf{0}]\!] & ,l[\![\mathbf{0}]\!]
\end{array}
\right\}
$$

$$\mathcal{R}_i^{Id}(x,y) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \mathsf{Q}_i(x,y) \,|\, \mathsf{R}_i & ,\mathsf{Q}_i(x,y) \,|\, \mathsf{R}_i \\ k_i[\![d!\langle x,y,l\rangle]\!] \,|\, \mathsf{R}_i & ,k_i[\![d!\langle x,y,l\rangle]\!] \,|\, \mathsf{R}_i \\ k_i[\![\mathsf{go}\, l.y!\langle f(x)\rangle]\!] & ,k_i[\![\mathsf{go}\, l.y!\langle f(x)\rangle]\!] \\ l[\![y!\langle f(x)\rangle]\!] & ,l[\![y!\langle f(x)\rangle]\!] \\ l[\![\mathbf{0}]\!] & ,l[\![\mathbf{0}]\!] \end{array} \right\}$$

where d, s stand for *data* and *sync* respectively and

$$\begin{array}{rcl} \mathsf{S}(y) & \Leftarrow & s?(x).y!\langle x\rangle \\ \mathsf{Q}_i(x,y) & \Leftarrow & \mathsf{go}\, k_i.d!\langle x,y,l\rangle \\ \mathsf{R}_i & \Leftarrow & k_i[\![d?(x,y,z).\mathsf{go}\, z\, .y!\langle f(x)\rangle]\!] \end{array}$$

We consider a number of possible transitions in $\mathcal{R}'(x,y)$. Assume we are in a state

$$\Gamma \triangleright (\nu d,s)\left(\begin{array}{l} l[\![\mathsf{S}(y)]\!] \\ |\, M_1 \,|\, M_2 \end{array}\right) ,\Gamma \triangleright \begin{array}{l} (\nu d,s)\left(\begin{array}{l} l[\![\mathsf{S}(y)]\!] \\ |\, N_1 \,|\, N_2 \end{array}\right) \\ |\, k_1[\![\mathsf{kill}]\!] \end{array} \quad \begin{array}{l} \text{where} \\ \langle M_1, N_1\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array}$$

then if we accept an answer from any replica and $l[\![\mathsf{S}(y)]\!]$ goes to $l[\![y!\langle f(x)\rangle]\!]$, then we transition to the state

$$\Gamma \triangleright (\nu d,s)\left(\begin{array}{l} l[\![y!\langle f(x)\rangle]\!] \\ |\, M_1' \,|\, M_2' \end{array}\right) ,\Gamma \triangleright (\nu d,s)\left(\begin{array}{l} l[\![y!\langle f(x)\rangle]\!] \\ |\, N_1' \,|\, N_2' \end{array}\right) \quad \begin{array}{l} \text{where} \\ \langle M_1', N_1'\rangle \in \mathcal{R}_1^{Id}(x,s) \\ \langle M_2', N_2'\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array}$$

If on the other hand, $k_1[\![\mathsf{kill}]\!]$ injects the fault, we transition to the pair

$$\Gamma \triangleright (\nu d,s)\left(\begin{array}{l} l[\![\mathsf{S}(y)]\!] \\ |\, M_1 \,|\, M_2 \end{array}\right) ,\Gamma - k_1 \triangleright (\nu d,s)\left(\begin{array}{l} l[\![\mathsf{S}(y)]\!] \\ |\, N_1 \,|\, N_2 \end{array}\right) \quad \begin{array}{l} \text{where} \\ \langle M_1, N_1\rangle \in \mathcal{R}_1^{\mathbf{0}}(x,s) \\ \langle M_2, N_2\rangle \in \mathcal{R}_2^{Id}(x,s) \end{array}$$

At this point, any actions by M_2 or N_2 are mapped by the identical action on the opposite side, while still remaining in the same state of the relation. If M_1 is involved in an action, then we have two cases: if the action involving M_1 causes $l[\![\mathsf{S}(y)]\!]$ to reduce to $l[\![y!\langle f(x)\rangle]\!]$ while reducing to M_1' itself, then we transition to

$$\Gamma \triangleright (\nu d,s)\left(\begin{array}{l} l[\![y!\langle f(x)\rangle]\!] \\ |\, M_1' \,|\, M_2 \end{array}\right) ,\Gamma - k_1 \triangleright (\nu d,s)\left(\begin{array}{l} l[\![y!\langle f(x)\rangle]\!] \\ |\, N_1 \,|\, N_2' \end{array}\right) \quad \begin{array}{l} \text{where} \\ \langle M_1', N_1\rangle \in \mathcal{R}_1^{\mathbf{0}}(x,s) \\ \langle M_2, N_2'\rangle \in \mathcal{R}_2^{\geq}(x,s) \end{array}$$

where on the right hand side N_2 has to reduce to N_2 while interacting with its respective $l[\![\mathsf{S}(y)]\!]$ using a weak action, to reduce it to $l[\![y!\langle f(x)\rangle]\!]$. We note that this internal interaction cannot be done by N_1 since k_1 is dead. Otherwise, if $l[\![\mathsf{S}(y)]\!]$ is not affected, we match the silent move from M_1 with the empty move. ■

Example 4.3.7 indicates that dynamic fault tolerance is stricter that static fault tolerance. In fact, a subset of the 1-static fault tolerant configurations in Example 4.3.5 satisfy

Table 24. *Network Operational Ruled for typed DπLoc*

$$\text{(l-fail)}\quad \frac{}{\Gamma \triangleright N \overset{\text{fail}}{\longmapsto} (\Gamma - l) \triangleright N}\ \Gamma \vdash l : \mathtt{c[a]}$$

the requirements of 1-dynamic fault tolerance. In spite of the indication that it is weaker, we argue, at least informally, that static fault tolerance is still useful for applications where fault tolerant conditions are not as stringent: to ensure dynamic fault tolerance, the additional fault tolerant mechanism required in terms of redundancy is often more expensive and in settings where there are limits on the number of resources that can be used, static fault tolerance might suffice.

4.4 Bisimulation techniques for Fault Tolerance

The fault tolerant definitions 4.3.4 (static) and 4.3.6 (dynamic), quantify over all valid fault contexts and even though we have tractable ways how to calculate fault tolerance through the bisimulations developed in § 4.2.2, we still have to go through a lot of unnecessary repeated work. For instance, to prove that server_3 is 2-dynamic fault tolerant, we need to provide 6 bisimulations, one of every different valid fault context; close inspection of these relations shows that there is a lot of overlap between the individual bisimulations.

Such an overlap would be automatically eliminated if we require a single relation that is somewhat the merging of all of these separate relations. Because of this reason, we reformulate our fault tolerant definitions to reflect such a merging of relations. In the sequel, we focus only on dynamic fault tolerance, since it is the most demanding of the two fault tolerance definitions and because we use it again in Chapter 5; a similar definition for the static case should not be more difficult to construct.

The new definition for dynamic fault tolerance is based on the failure-free actions defined earlier in § 4.2.2 together with the new action defined in Table 24. Intuitively, the new action fail allows the observer to kill confined locations but prohibits it from determining which confined location died. Stated otherwise, the action allows the observer to *count* the number of confined locations that are killed. Counting of confined location killings is central to Definition 4.4.1: The asymmetric relation defined, $\leq_D^n$, is parameterised with a number n, denoting the maximum number of confined locations that can be killed on the right hand side. Thus, a fail move on the right hand side may be matched by a weak τ-move on the left hand side and the two residuals need to be related in $\leq_D^{n-1}$.

Definition 4.4.1 (Dynamic Fault Tolerant Simulation). Dynamic n-fault tolerant *simulation*, denoted $\leq_D^n$, is the largest *asymmetric* typed relation over configurations such that whenever $\Gamma_1 \triangleright M_1 \leq_D^n \Gamma_2 \triangleright M_2$,

- $\Gamma_1 \triangleright M_1 \overset{\gamma}{\longmapsto} \Gamma_1' \triangleright M_1'$ implies $\Gamma_2 \triangleright M_2 \overset{\widehat{\gamma}}{\Longmapsto} \Gamma_2' \triangleright M_2'$ such that $\Gamma_1' \triangleright M_1' \leq_D^n \Gamma_2' \triangleright M_2'$

- $\Gamma_2 \triangleright M_2 \overset{\gamma}{\longmapsto} \Gamma_2' \triangleright M_2'$ implies $\Gamma_1 \triangleright M_1 \overset{\widehat{\gamma}}{\Longmapsto} \Gamma_1' \triangleright M_1'$ such that $\Gamma_1' \triangleright N_1' \preceq_D^n \Gamma_2' \triangleright M_2'$
- if $n > 0$, $\Gamma_2 \triangleright M_2 \overset{\mathsf{fail}}{\longmapsto} \Gamma_2' \triangleright M_2'$ implies $\Gamma_1 \triangleright M_1 \Longmapsto \Gamma_1' \triangleright M_1'$ such that $\Gamma_1' \triangleright M_1' \preceq_D^{n-1} \Gamma_2' \triangleright M_2'$

■

Before we can use Definition 4.4.1 to show that a configuration is dynamic fault tolerant according to Definition 4.3.6, we need to show that, the former definition is sound with respect to the latter. To show this, we prove the following lemmas.

Lemma 4.4.2 (Internal and External Kills). If $\Gamma \triangleright M \,|\, l[\![\mathsf{kill}]\!] \overset{\tau}{\longmapsto} \Gamma - l \triangleright M \,|\, l[\![\mathbf{0}]\!]$ then

- If $\Gamma \vdash l : \mathsf{p}$, then $\Gamma \triangleright M \overset{\mathsf{kill}:l}{\longmapsto} \Gamma - l \triangleright M$
- If $\Gamma \vdash l : \mathsf{c}$, then $\Gamma \triangleright M \overset{\mathsf{fail}}{\longmapsto} \Gamma - l \triangleright M$

Proof. From

$$\Gamma \triangleright M \,|\, l[\![\mathsf{kill}]\!] \overset{\tau}{\longmapsto} \Gamma - l \triangleright M \,|\, l[\![\mathbf{0}]\!]$$

we know $\Gamma \vdash l : \textbf{alive}$. Thus if $\Gamma \vdash l : \mathsf{p}$, we combine these to get $\Gamma \vdash l : \mathsf{p}[\mathsf{a}]$ and by (l-halt) we get

$$\Gamma \triangleright M \overset{\mathsf{kill}:l}{\longmapsto} \Gamma - l \triangleright M$$

otherwise we can combine $\Gamma \vdash l : \textbf{alive}$ and $\Gamma \vdash l : \mathsf{c}$ to obtain $\Gamma \vdash l : \mathsf{c}[\mathsf{a}]$ and by (l-fail) we get

$$\Gamma \triangleright M \overset{\mathsf{fail}}{\longmapsto} \Gamma - l \triangleright M$$

as required. □

Proposition 4.4.3 (Soundness of $\preceq_D^n$).

$$\Gamma \models M_1 \;\preceq_D^n\; M_2 \text{ implies } \forall F^n \text{ such that } \Gamma \vdash F^n \text{ then } \Gamma \models M_1 \;\cong_{ff}\; M_2 \,|\, F^n$$

Proof. Let $\mathcal{R}_n$ be a relation parameterised by a number n and defined as

$$\mathcal{R}_n \overset{\text{def}}{=} \left\{ \Gamma_1 \triangleright M_1 \quad, \Gamma_2 \triangleright M_2 \,|\, F^i \quad \middle|\; \mathcal{I} \models \Gamma_1 \triangleright M_1 \;\preceq_D^i\; \Gamma_2 \triangleright M_2,\ \mathcal{I} \vdash F^i \text{ and } 0 \leq i \leq n \right\}$$

We prove that $\preceq_D^n$ is sound with respect to n-dynamic fault tolerance by showing that $\mathcal{R}_n$ is a failure-free bisimulation, that is $\mathcal{R}_n \subseteq \approx_{ff}$.

We focus on the actions possible by the right hand side, and leave the simpler case, that is the actions possible by the left hand side, to the interested reader. Thus assume $\Gamma_1 \triangleright M_1 \;\mathcal{R}_n\; \Gamma_2 \triangleright M_2 \,|\, F^i$ for some $0 \leq i \leq n$ and we have

$$\Gamma_2 \triangleright M_2 \,|\, F^i \overset{\gamma}{\longmapsto} \Gamma_2' \triangleright M_2'$$

We have to show that

$$\Gamma_1 \triangleright M_1 \overset{\widehat{\gamma}}{\Longmapsto} \Gamma_1' \triangleright M_1' \text{ such that } \Gamma_1' \triangleright M_1' \;\mathcal{R}_n\; \Gamma_2' \triangleright M_2'$$

From the structure of F^i, we deduce that there can be no interaction between M_2 and F^i and by (l-par) we conclude that this action can be caused by either of the following actions

$$\text{either } \Gamma_2 \triangleright M_2 \overset{\gamma}{\longmapsto} \Gamma_2' \triangleright M_2'' \text{ where } M_2' \equiv M_2'' \,|\, F^i \tag{4.39}$$

$$\text{or } \Gamma_2 \triangleright F^i \overset{\gamma}{\longmapsto} \Gamma_2' \triangleright F' \text{ where } M_2' \equiv M_2 \,|\, F' \tag{4.40}$$

If the action is caused by (4.39), then from the end clause of definition of $\mathcal{R}_n$ and the definition of $\preceq_D^i$ we know that (4.39) can be matched by

$$\Gamma_1 \triangleright M_1 \overset{\widehat{\gamma}}{\Longmapsto} \Gamma_1' \triangleright M_1' \text{ where } \Gamma_1' \triangleright M_1' \preceq_D^i \Gamma_2' \triangleright M_2''$$

and as a result we know that $\Gamma_1' \triangleright M_1' \mathcal{R}_n \Gamma_2' \triangleright M_2'' \,|\, F^i$.

If on the other hand, the action is caused by (4.40), then from the structure of F^i, we conclude that γ can only have the form τ. From the assumption that $\Gamma_2 \vdash F^i$, we can rewrite (4.40) as

$$\Gamma_2 \triangleright k_j[\![\mathsf{kill}]\!] \,|\, F^{i-1} \overset{\tau}{\longmapsto} \Gamma_2 - k_j \triangleright k_j[\![\mathbf{0}]\!] \,|\, F^{i-1} \text{ for some } k_j \text{ such that } \Gamma_2 \vdash k_j : \mathsf{c} \tag{4.41}$$

From (4.41) and the application of (l-par), we obtain

$$\Gamma_2 \triangleright M_2 \,|\, k_j[\![\mathsf{kill}]\!] \overset{\tau}{\longmapsto} \Gamma_2 - k_j \triangleright M_2 \,|\, k_j[\![\mathbf{0}]\!] \tag{4.42}$$

and by (4.42) and Lemma 4.4.2 we get

$$\Gamma_2 \triangleright M_2 \overset{\mathsf{fail}}{\longmapsto} \Gamma_2 - k_j \triangleright M_2 \tag{4.43}$$

By (4.43), the final clause of definition of $\mathcal{R}_n$ and $\preceq_D^i$ we get the matching move

$$\Gamma_1 \triangleright M_1 \Longmapsto \Gamma_1' \triangleright M_1' \text{ where } \Gamma_1' \triangleright M_1' \preceq_D^{i-1} \Gamma_2 - k_j \triangleright M_2$$

from which we deduce

$$\Gamma_1' \triangleright M_1' \;\mathcal{R}_n\; \Gamma_2 - k_j \triangleright M_2|F^{i-1}$$

as required. □

Corollary 4.4.4. If $\Gamma \models M \preceq_D^n M$ then $\Gamma \triangleright M$ is n-dynamically fault tolerant.

Proof. Immediate consequence of Proposition 4.4.3 and Definition 4.3.6 □

Any proofs for fault tolerance would greatly benefit if, in addition to $\preceq_D^n$, we apply the same techniques for β-moves and structural equivalence developed in § 4.2.3 to $\preceq_D^n$, so that we only need to provide fault tolerant simulations up-to β-moves and structural equivalence. We thus dedicate the remainder of the chapter to develop up-to techniques for $\preceq_D^n$.

We start by giving a definition of a *counting* bisimulation, that can match the number of confined location killings and compare behaviour after these killings happen. Admittedly, this bisimulation is hard to justify on it own; we however use it to facilitate the development of up-to-β-move techniques for $\preceq_D^n$.

Definition 4.4.5 (Counting weak bisimulation equivalence). We let ρ range over the actions $(\tilde{n}:\tilde{\mathsf{S}})l : a!\langle V\rangle$, $(\tilde{n}:\tilde{\mathsf{S}})l : a?(V)$, τ and the new action fail. Then, counting weak bisimulation equivalence, denoted as $\approx_{cnt}$, is defined to be the largest typed relation over configurations such that if $\Gamma_1 \triangleright M_1 \approx_{cnt} \Gamma_2 \triangleright M_2$ then

- $\Gamma_1 \triangleright M_1 \overset{\rho}{\longmapsto} \Gamma_1' \triangleright M_1'$ implies $\Gamma_2 \triangleright M_2 \overset{\hat{\rho}}{\Longmapsto} \Gamma_2' \triangleright M_2'$ such that $\Gamma_1' \triangleright M_1' \approx_{cnt} \Gamma_2' \triangleright M_2'$
- $\Gamma_2 \triangleright M_2 \overset{\rho}{\longmapsto} \Gamma_2' \triangleright M_2'$ implies $\Gamma_1 \triangleright M_1 \overset{\hat{\rho}}{\Longmapsto} \Gamma_1' \triangleright M_1'$ such that $\Gamma_1' \triangleright M_1' \approx_{cnt} \Gamma_2' \triangleright M_2'$ ∎

We next show that the commutative (diamond) property of β-moves is preserved for ρ actions, up-to configurations that are $\equiv_f$ structurally equivalent.

Lemma 4.4.6. Suppose $\Gamma \triangleright N \overset{\tau}{\longmapsto}_\beta \Gamma \triangleright M$. Then for every action $\Gamma \triangleright N \overset{\rho}{\longmapsto} \Gamma' \triangleright N'$,

- either $\mu = \tau$ and $\Gamma \triangleright M = \Gamma' \triangleright N'$
- or there is a configuration $\Gamma' \triangleright M'$ such that $\Gamma' \triangleright N' \Longmapsto_\beta \equiv_f \Gamma' \triangleright M'$ and $\Gamma \triangleright M \overset{\rho}{\longmapsto} \Gamma' \triangleright M'$.

Proof. The proof is similar to the one for Lemma 4.2.28, by case analysis on ρ and induction on the length of derivation of $\Gamma \triangleright N \overset{\tau}{\longmapsto}_\beta \Gamma \triangleright M$; we here consider the only different case, $\rho = \mathsf{fail}$, as an example.

If $\rho = \mathsf{fail}$ then through a lemma such as Lemma 3.5.3, we deduce that

$$\begin{aligned} &\Gamma \vdash l : \mathsf{c}[\mathsf{a}] \text{ where} \\ &\Gamma \triangleright N \overset{\mathsf{fail}}{\longmapsto} \Gamma - l \triangleright N \end{aligned} \tag{4.44}$$

By induction on the length of derivation, if the last rule used in the derivation of $\Gamma \triangleright N \overset{\tau}{\longmapsto}_\beta \Gamma \triangleright M$ is (b-eq), then we can easily deduce that N and M have the form $l[\![\mathsf{if}\ v = v.P\lceil P\rceil]\!]$ and $l[\![P]\!]$ respectively.

Thus the weak beta move required is matched by the empty move and from (4.44) and (l-fail) we deduce

$$\Gamma \triangleright l[\![P]\!] \overset{\mathsf{fail}}{\longmapsto} \Gamma - l \triangleright l[\![P]\!]$$

and it is easy to show that

$$\Gamma - l \triangleright l[\![\mathsf{if}\ v = v.P\lceil P\rceil]\!] \equiv_f \Gamma - l \triangleright l[\![P]\!]$$

□

Before we give the actual definition of our n-fault tolerant simulation up-to-β-moves and justify it, we still need to prove these two lemmas to stitch up the proof.

Proposition 4.4.7 (β-Structural equivalence and Counting bisimulation). If $\Gamma \models M \equiv_f N$ then $\Gamma \models M \approx_{cnt} N$

Proof. By simple case analysis of why $M \equiv_f N$. □

Proposition 4.4.8. Suppose $\Gamma \triangleright N \Longmapsto_\beta \Gamma \triangleright M$. Then $\Gamma \models N \approx_{cnt} M$.

Proof. An immediate consequence of Lemma 4.4.6. □

Definition 4.4.9 (Fault Tolerant Simulation up-to β-moves). An n-fault tolerant simulation up-to β-moves, denoted as $\leq^n_\beta$, is the largest typed relation $\mathcal{R}$ between configurations parameterised by the number n, such that whenever we have $\Gamma_1 \triangleright M_1 \leq^n_\beta \Gamma_2 \triangleright M_2$ then

- $\Gamma_1 \triangleright M_1 \overset{\gamma}{\longmapsto} \Gamma'_1 \triangleright M'_1$ implies $\Gamma_2 \triangleright M_2 \overset{\hat{\gamma}}{\Longmapsto} \Gamma'_2 \triangleright M'_2$ such that $\Gamma'_1 \triangleright M'_1 \mathcal{A}_l \circ \leq^n_\beta \circ \approx_{cnt} \Gamma'_2 \triangleright M'_2$
- $\Gamma_2 \triangleright M_2 \overset{\gamma}{\longmapsto} \Gamma'_2 \triangleright M'_2$ implies $\Gamma_1 \triangleright M_1 \overset{\hat{\gamma}}{\Longmapsto} \Gamma'_1 \triangleright M'_1$ such that $\Gamma'_2 \triangleright M'_2 \mathcal{A}_l \circ \leq^n_\beta \circ \approx_{ff} \Gamma'_1 \triangleright M'_1$
- If $n > 0$ then $\Gamma_2 \triangleright M_2 \overset{\mathsf{fail}}{\longmapsto} \Gamma'_2 \triangleright M'_2$ implies $\Gamma_1 \triangleright M_1 \Longmapsto \Gamma'_1 \triangleright M'_1$ such that $\Gamma'_2 \triangleright M'_2 \leq^{n-1}_\beta \circ \approx_{ff} \Gamma'_1 \triangleright M'_1$

where $\mathcal{A}_l$ is the relation $\Longmapsto_\beta \circ \equiv_f$. ■

We highlight the asynchrony in the above definition, where for the first clause we use the new counting bisimulation, $\approx_{cnt}$, for matching the residuals whereas for the second and third clauses we use $\approx_{ff}$ as before. The reason for this is that the right hand side configuration can also produce fail moves if $n \geq 1$, which we have to match using $\approx_{cnt}$. We use this fact in the proofs below.

Lemma 4.4.10. Suppose $\Gamma_1 \triangleright M_1 \leq^n_\beta \Gamma_2 \triangleright M_2$ then

- if $\Gamma_1 \triangleright M_1 \overset{\widehat{\mu}}{\Longmapsto} \Gamma'_1 \triangleright M'_1$ then $\Gamma_2 \triangleright M_2 \overset{\widehat{\mu}}{\Longmapsto} \Gamma'_2 \triangleright M'_2$, such that $\Gamma'_1 \triangleright M'_1 \approx \circ \approx_\beta \circ \approx_{cnt} \Gamma'_2 \triangleright M'_2$
- if $\Gamma_2 \triangleright M_2 \overset{\widehat{\mu}}{\Longmapsto} \Gamma'_2 \triangleright M'_2$ then $\Gamma_1 \triangleright M_1 \overset{\widehat{\mu}}{\Longmapsto} \Gamma'_1 \triangleright M'_1$, such that $\Gamma'_1 \triangleright M'_1 \approx \circ \approx_\beta \circ \approx_{cnt} \Gamma'_2 \triangleright M'_2$
- if $n \geq 1$ and $\Gamma_2 \triangleright M_2 \overset{\mathsf{fail}}{\Longmapsto} \Gamma'_2 \triangleright M'_2$ then $\Gamma_1 \triangleright M_1 \Longmapsto \Gamma'_1 \triangleright M'_1$, such that $\Gamma'_1 \triangleright M'_1 \approx \circ \approx_\beta \circ \approx_{cnt} \Gamma'_2 \triangleright M'_2$

Proof. Similar to the earlier proof for Lemma 4.2.31, where we have to complete a series of matching moves using Lemma 4.4.6, Proposition 4.4.7 and Proposition 4.4.8. □

Proposition 4.4.11 (Inclusion of fault tolerant simulation up-to β-moves). If $\Gamma_1 \triangleright M_1 \leq^n_\beta \Gamma_2 \triangleright M_2$ then $\Gamma_1 \triangleright M_1 \leq^n_D \Gamma_2 \triangleright M_2$

Proof. We prove the above proposition by defining the relation $\mathcal{R}_n$ as

$$\mathcal{R}_n \quad = \quad \left\{ \ \Gamma_1 \triangleright M_1 \quad , \Gamma_2 \triangleright M_2 \quad \middle| \ \Gamma_1 \triangleright M_1 \approx_{ff} \circ \leq^i_\beta \circ \approx_{cnt} \Gamma_2 \triangleright M_2 \text{ and } 0 \leq i \leq n \ \right\}$$

and show that $\mathcal{R}_n \subseteq \leq^n_D$. The required result can then be extracted from this result by considering the special cases where $\approx_{ff}$ and $\approx_{cnt}$ on either side are the identity relations.

The interesting case in this proof is when $\Gamma_2 \triangleright M_2 \overset{\mathsf{fail}}{\longrightarrow} \Gamma'''_2 \triangleright M'''_2$, since we exploit the use of $\approx_{cnt}$ in Definition 4.4.9.

By Definition of $\mathcal{R}_n$, $\exists M_1''$ and $\Gamma_2' \triangleright M_2'$ such that

$$\Gamma_1 M_1 \approx_{ff} \Gamma_1'' M_1'' \tag{4.45}$$
$$\Gamma_1'' \triangleright M_1'' \;\leq_\beta^i\; \Gamma_2' \triangleright M_2' \tag{4.46}$$
$$\Gamma_2' \triangleright M_2' \;\approx_{cnt}\; \Gamma_2 \triangleright M_2 \tag{4.47}$$

By (4.47) we derive

$$\Gamma_2' \triangleright M_2' \stackrel{\text{fail}}{\Longrightarrow} \Gamma_2'' \triangleright M_2'' \;\text{ such that } \Gamma_2'' \triangleright M_2'' \approx_{cnt} \Gamma_2''' \triangleright M_2''' \tag{4.48}$$

By Lemma 4.4.10, (4.48), (4.46) and the definition of $\leq_\beta^i$ for some $1 \leq i \leq n$ we deduce

$$\Gamma_1'' \triangleright M_1'' \stackrel{\widehat{\tau}}{\Longrightarrow} \Gamma_1''' \triangleright M_1''' \;\text{ such that } \Gamma_1''' \triangleright M_1''' \approx_{ff} \circ \leq_\beta^{i-1} \circ \approx_{cnt} \Gamma_2'' \triangleright M_2'' \tag{4.49}$$

By induction on the number of actions in (4.49) and (4.45) we get

$$\Gamma_1 \triangleright M_1 \stackrel{\widehat{\mu}}{\Longrightarrow} \Gamma_1' \triangleright M_1' \;\text{ such that } \Gamma_1''' \triangleright M_1''' \approx_{ff} \Gamma_1' \triangleright M_1' \tag{4.50}$$

which is our matching move. Once again, from (4.48), (4.49), (4.50) and the transitivity of $\approx_{cnt}$ and $\approx_{ff}$ we get

$$\Gamma_1' \triangleright M_1' \approx_{ff} \circ \leq_\beta^{i-1} \circ \approx_{cnt} \Gamma_2''' \triangleright M_2'''$$

which is still in $\mathcal{R}_n$. □

Example 4.4.12. [Fault tolerant Simulation] Using the result of Proposition 4.4.3 and Proposition 4.4.7 we can now prove that server_1 is 1-dynamically fault tolerant by giving the *single* relation

$$\mathcal{R} \stackrel{\text{def}}{=} \{\langle \Gamma \triangleright \mathsf{server}_2, \Gamma \triangleright \mathsf{server}_2 \rangle\} \cup \left(\bigcup_{u,v \in \textsc{Names}} \mathcal{R}'(u,v) \right)$$

as opposed to the three relations required in Example 4.3.7. Once again, $\mathcal{R}$ is the union of all the relations $\mathcal{R}'(x,y)$ where the variables x, y are parameterised by names $u, v \in \textsc{Names}$. We also reuse the relations $\mathcal{R}_i^{Id}(x,y)$,$\mathcal{R}_i^{\mathbf{0}}(x,y)$, $\mathcal{R}_i^{\geq}(x,y)$, R_i, $\mathsf{Q}_i(x,y)$ and $\mathsf{S}(y)$ defined earlier in Example 4.3.7.

$\mathcal{R}'(x,y)$ is a generalization of the previous relation given in Example 4.3.7 where we now consider up to one fault injected at either location k_1, k_2 and k_3. The case where k_3 dies is similar to the case when no fault is injected because server_2 does not have any located processes dependent on k_3. The case for when k_1 is identical to Example 4.3.7 whereas the case where k_2 dies is dual with k_1.

$$
\mathcal{R}'(x,y) \stackrel{\text{def}}{=} \left\{
\begin{array}{lll|l}
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma - k_1 \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\geq}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma - k_2 \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{\geq}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma - k_3 \triangleright (\nu d,s)\begin{pmatrix} l\llbracket \mathsf{S}(y) \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \\
\\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma - k_1 \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\mathbf{0}}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{\geq}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma - k_2 \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\geq}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{\mathbf{0}}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,M_1\,|\,M_2 \end{pmatrix} & , \Gamma - k_3 \triangleright (\nu d,s)\begin{pmatrix} l\llbracket y!\langle f(x)\rangle \rrbracket \\ |\,N_1\,|\,N_2 \end{pmatrix} & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \\
\\
\Gamma \triangleright (\nu d,s)\big(\, M_1\,|\,M_2 \,\big) & , \Gamma \triangleright (\nu d,s)\big(\, N_1\,|\,N_2 \,\big) & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\big(\, M_1\,|\,M_2 \,\big) & , \Gamma - k_1 \triangleright (\nu d,s)\big(\, N_1\,|\,N_2 \,\big) & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\mathbf{0}}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{\geq}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\big(\, M_1\,|\,M_2 \,\big) & , \Gamma - k_2 \triangleright (\nu d,s)\big(\, N_1\,|\,N_2 \,\big) & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{\geq}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{\mathbf{0}}_2(x,s) \end{array} \\
\Gamma \triangleright (\nu d,s)\big(\, M_1\,|\,M_2 \,\big) & , \Gamma - k_3 \triangleright (\nu d,s)\big(\, N_1\,|\,N_2 \,\big) & & \begin{array}{l} \langle M_1,N_1\rangle \in \mathcal{R}^{Id}_1(x,s) \\ \langle M_2,N_2\rangle \in \mathcal{R}^{Id}_2(x,s) \end{array}
\end{array}
\right\}
$$

■

4.5 Summary

In this Chapter we formalised a definition of fault-tolerance for DπLoc, an extension of Dπ with location failure *only*: this fault-tolerance definition is paramertised by a natural number n, denoting the maximum number of faults that are assumed to occur. Our definition was based on a partitioning of resources into two categories, dependable and non-dependable: the observations are limited to dependable resources and fault are injected on the non-dependable resources.

We developed a type-system, which ensured that an observer limited to dependable resources never gains access to non-dependable resources, and obtained *type safety* and *subject reduction* results for this type system. We also refined the bisimulation developed in Chapter 2 and proved that it is *fully abstract* with respect to a reduction barbed congruence that assumes observers limited to dependable resources. We then showed how this bisimulation can be used to prove n-fault tolerance for certain DπLoc configurations.

We then went a step further and developed what we called fault tolerant simulations, adapting up-to techniques based on β-moves, [JR04], to enable a better handling of our proof for fault tolerance, since, even for the simple fault-tolerant configurations considered in this Chapter, the bisimulations required using our preliminary methods were of a prohibitive size. We obtained soundness results for these techniques.

To the best of our knowledge, this is the first time fault-tolerance has been investigated in a distributed calculus setting. We postulate that the fault tolerance definition we formalised is general enough to be applied to fault-tolerant systems that encode a notion of state; the examples we considered in this Chapter used stateless replication. This would enable us to study a wider spectrum of fault-tolerance techniques, such as those using lazy replication, discussed in the Introduction. The bisimulation techniques we developed should also suffice for the necessary proofs.

Chapter 5

Case Study: Consensus

It has long been understood that systems can enhance their level of dependability through fault tolerance. In his survey [Fis83], Fischer claims that consensus is a pervasive *fault tolerant* problem in distributed computing. Despite of this claim, even though consensus has been throughly studied, solved and proved to be correct under a number of assumptions [CT96], to the best of our knowledge, it has never be approach from a fault tolerance perspective.

In this Chapter, we explore the viability of proving the correctness of a consensus solving algorithm by viewing it as fault tolerance problem; we consider the simplest scenario for consensus where perfect failure detectors are used [CT96], implement the algorithm in typed DπLoc, and then prove its correctness using the theory developed in Chapter 4. Encoding the pseudo-code algorithm given in [CT96] in typed DπLoc ensures a formal development in a well defined language and as a result, we can use the corresponding typed DπLoc theory. Moreover, such an encoding would be an opportunity to demonstrate the expressivity of typed DπLoc.

5.1 Consensus Overview

The *specification* of the consensus problem is based on n processes that execute *independently* from one another. The task of every process is to *input* a value v from a set of values V and then *decide* by outputting a value $v' \in V$. At any point during the execution, the process may fail independently from other processes in *fail-stop fashion*. The critical aspect of consensus specification is that three correctness conditions need to be satisfied:

Termination: All non-failing processes must eventually decide.

Agreement: No two processes decide on different values

Validity: If all processes are given the same value $v \in V$ as input, then v is the only possible decision value.

The implication of the last two conditions is that every process needs to coordinate with other processes in order to agree on a value before deciding: since the value decided

is dependent on the input (validity), such a coordination cannot be avoided by pre-agreeing on some default value before the execution starts. The main task is to guarantee termination, agreement and validity, irrespective of the failures that occur at any point during execution.

The seminal paper by Toueg and Chandra, [CT96], classifies *consensus-solving algorithms* according to the type of failure detectors used. They also give algorithms in pseudocode for a number of these classes and prove their correctness using an *algorithmic analysis*. We here adapt the simplest of the algorithms given in [CT96] using perfect failure detectors. This algorithm assumes n *asynchronous, replicated* participants, where every participant, P_i, holds an initial value v from a value set V. Each participant tries to attain consensus in flood-set fashion, by:

1. broadcasting its value to *all* the other participants.
2. receiving enough values from all the other participants to be able to decide on a value.

Intuitively, since all participants are replicas, they use the *same decision criteria* and assuming that they all receive the same (flood) set of value on which to base their decisions, the participants should decide the same value, thereby attaining consensus. The complication however arises because participants may *fail* at any point in time; this may have two adverse effects on the algorithm, namely:

Decision Blocking: a participant may be waiting for a value to arrive from another participant which has failed before deciding. In this case, the value will never reach the waiting participant, which will block and never decide.

Corrupted Broadcast: a participant may broadcast its values to a *subset* of the other participants before failing. This may lead to an incomplete set of values at every participant.

The algorithm is thus made *fault tolerant* in two ways:

1. The problem of blocking is solved through the use of perfect failure detectors: every participant P_i employs $(n-1)$ failure detectors, each observing the state of the remaining $(n-1)$ participants. Once a participant P_j fails, the failure detector for P_j at P_i instructs P_i not to wait for the value from P_j.
2. The problem of corrupted broadcast is overcome by using consecutive rounds of broadcasting, gathering sets of values (ignoring the values of dead participants) and updating the decided value. If the *maximum* number of participants failing is m, then by performing $m + 1$ rounds, we are guaranteed that there is at least one round with no corrupted broadcast, and as a result, all the participants will reach consensus at that round. Moreover, for the remaining rounds, participants will be broadcasting to each other the agreed estimate, and since the decision criteria at every participant needs to satisfy validity, every participant holds that value for the remaining rounds.

5.2 Defining Consensus in typed DπLoc

In this section, we adapt the definition of consensus over-viewed in § 5.1 to a specific DπLoc setting. We precede our adaptation of the consensus problem by stating a number of assumptions about the network over which the system solving consensus executes. More specifically, we assume that to solve consensus for n processes, the underlying network consists of at least n locations, $l_1, \ldots, l_n$; this allows us to distribute the participants $\mathsf{P}_{1..n}$ at distinct locations, thereby simulating *n independently failing* processes. In addition, we assume that these locations are *confined* so as to limit the observation of their failure and facilitate our analysis.

For the purposes of our study, the *decision set* is chosen to be the minimum possible, that is the set of boolean values $V = \{t, f\}$. We model these values by two free channel names, named t and f respectively. We also assume that the network representation permits every participant to use the free channel p to input the *proposed* initial value from the value set and the free channel d to return the *decided* value at the end of the n rounds. Finally, we assume that the network representation defines a public channel $\textsc{succ}$ and a public location k_0 that are fresh to the algorithm solving consensus. We formalise all this in the definition below, using the abbreviation notation defined earlier in § 4.2.3.

Definition 5.2.1 (Consensus Permitting Network Representation). For any system N, a network representation Γ is *consensus permitting* for n participants iff

1. $\forall i = 1..n$ we have $\Gamma \vdash l_i : \mathtt{c[a]}$ (The network representation specifies n confined locations where to distribute our participants.
2. $\Gamma \vdash t : \mathtt{ch}\langle - \rangle,\, f : \mathtt{ch}\langle - \rangle,\, p : \mathtt{ch}\langle \mathtt{ch}\langle - \rangle \rangle,\, d : \mathtt{ch}\langle \mathtt{ch}\langle - \rangle \rangle$ (The network representation specifies two channels t, f representing the every value in the value set V and another two channels, one to initialise the participant with a value in the value set, p, and the other to obtain the decided value, d)
3. $\Gamma \vdash \textsc{succ} : \mathtt{p}\langle\rangle,\, k_0 : \mathtt{p[a]}$ where $\textsc{succ}, k_0 \notin \mathbf{fn}(N)$ (The network representation defines a public channel, $\textsc{succ}$ and a public location k_0, both fresh to N)
4. $\Gamma \vdash N$ (The configuration $\Gamma \triangleright N$ is well-formed) ■

We next specify the three requirements defining consensus for DπLoc systems, given earlier in § 5.1. The definitions of termination, validity and agreement are based on the encodings just stated, and formalised in Definition 5.2.1. Before we do this, we adapt a standard notion of termination used in process calculi and call it *stabilisation* to avoid confusion with the first criteria of consensus.

Definition 5.2.2 (Stabilisation). A *stable* configuration is a configuration $\Gamma \triangleright N$ that can not perform any reduction. This is denoted as

$$\Gamma \triangleright N \not\longrightarrow$$

Thus a configuration $\Gamma \triangleright N$ *eventually stabilises*, (or stabilise for short) if there exists a maximal number of reduction steps n after which it always stops reducing. More formally

$$\exists\, n \text{ such that } \forall \Gamma', N' \text{ if } \Gamma \triangleright N \underbrace{\longrightarrow \cdot \ldots \cdot \longrightarrow}_{\leq n} \Gamma' \triangleright N' \text{ then } \Gamma' \triangleright N' \not\longrightarrow$$

■

We here define two forms of termination, called *weak termination* and *strong termination*. Intuitively, weak termination states that if a system N is initialised properly, then for all possible reductions, the system *can* return a decision at every location l_i. However, weak termination does not guarantee that the system *will* return a decision at every location l_i; indeed it can loop forever internally. For this stronger condition we use strong termination.

Definition 5.2.3 (Termination).

- A system N satisfies *weak termination* for n participants if and only if, there exists an n-participant consensus permitting network representation Γ for N such that when we initialize every participant in N with any a value $v \in V$, we are guaranteed that, for all reductions possible, all the n participants can eventually decide, or else die. More formally, for any $n-1$ dynamically fault inducing contexts, F^{n-1}, such that $\Gamma \vdash \mathsf{F}^{n-1}$, we have:

$$\begin{array}{l} \forall \Gamma', N' \\ \quad \text{if } \Gamma \triangleright N \,|\, \mathsf{F}^{n-1} \,|\, (\nu\, c_{1..n}) \left(\begin{array}{l} k_0[\![c_1?()...c_n?().\textsc{succ}!\langle\rangle]\!] \,| \\ \prod_{i=1}^{n} \left(\begin{array}{l} l_i[\![p!\langle t\rangle \,|\, p!\langle f\rangle]\!] \,| \\ l_i[\![d?(x).\mathsf{go}\; k_0.c_i!\langle\rangle]\!] \,| \\ k_0[\![\mathsf{monitor}\; l_i\lceil c_i!\langle\rangle\rceil]\!] \end{array} \right) \end{array} \right) \longrightarrow^{*} \Gamma' \triangleright N' \\ \quad \text{then } \Gamma' \triangleright N' \Downarrow_{\textsc{succ}@k_0} \end{array}$$

  In the above test we first initialise all participants with either t or f, $l_i[\![p!\langle t\rangle \,|\, p!\langle f\rangle]\!]$. We then define n scoped channels $c_{1..n}$, where c_i denotes completion for participant P_i. Completion can be triggered if either P_i decides, that is outputting a value on channel d at l_i, or else P_i dies: the code $l_i[\![d?(x).\mathsf{go}\; k_0.c_i!\langle\rangle]\!]$ and $k_0[\![\mathsf{monitor}\; l_i\lceil c_i!\langle\rangle\rceil]\!]$ check for either and report the signalling output on c_i at the fresh location k_0. Here, the code $k_0[\![c_1?()...c_n?().\textsc{succ}!\langle\rangle]\!]$ checks that every participant completes and reports on the fresh channel $\textsc{succ}$ when this happens.

- A system N satisfies *strong termination* for n participants if and only if, there exists an n-participant consensus permitting network representation Γ for N such that $\Gamma \triangleright N$

  1. weakly terminates
  2. stabilises ■

Definition 5.2.4 (Validity). A system N satisfies *validity* for n participants if and only if, there exists an n-participant consensus permitting network representation Γ for N such that when we initialize every participant in N with a homogeneous value $v \in V$, we are guaranteed that, for all reductions possible, no n participants will never decide anything else other than the value v. More formally, for any $n-1$ fault inducing contexts, F^{n-1}, such that $\Gamma \vdash \mathsf{F}^{n-1}$ we have:

$$\Gamma \triangleright N \,|\, \mathsf{F}^{n-1} \,|\, \prod_{i=1}^{n} l_i[\![p!\langle t\rangle]\!] \,|\, l_i[\![d?(x).\mathsf{if}\;(x=f).\mathsf{go}\; k_0.\textsc{succ}!\langle\rangle]\!] \not\Downarrow_{\textsc{succ}@k_0}$$

and

$$\Gamma \triangleright N \,|\, \mathsf{F}^{n-1} \,|\, \prod_{i=1}^{n} l_i[\![p!\langle f\rangle]\!] \,|\, l_i[\![d?(x).\mathsf{if}\;(x=t).\mathsf{go}\; k_0.\textsc{succ}!\langle\rangle]\!] \not\Downarrow_{\textsc{succ}@k_0}$$

In the above two tests, we initialise every participant with a homogeneous value: in the first case all participants are initialise with a t value using the code $l_i[\![p!\langle t\rangle]\!]$ whereas in the second case all participants are set to f using $l_i[\![p!\langle f\rangle]\!]$. In both tests we then collect decided values, if any, and check whether it is different from the value inputs, that is $l_i[\![d?(x).\mathsf{if}\ (x = f).\mathsf{go}\ k_0.\text{SUCC}!\langle\rangle]\!]$ for the first test and $l_i[\![d?(x).\mathsf{if}\ (x = t).\mathsf{go}\ k_0.\text{SUCC}!\langle\rangle]\!]$ for the second. If so, we report on the fresh channel SUCC at location k_0. ■

Definition 5.2.5 (Agreement). A system N satisfies *agreement* for n participants if and only if, there exists an n-participant consensus permitting network representation Γ for N such that, when we initialize every participant in N with a value $v \in V$, we are guaranteed that, for all reductions possible, no two participants will ever decide different values, $v', v'' \in V$ where $v' \neq v''$. More formally, for any $n-1$ fault inducing contexts, F^{n-1}, such that $\Gamma \vdash \mathsf{F}^{n-1}$ we have:

$$\Gamma \triangleright N \,|\, \mathsf{F}^{n-1} \,|\, (\nu\, c)\left(\begin{array}{l} k_0[\![c?(x).c?(y).\mathsf{if}\ (x \neq y).\text{SUCC}!\langle\rangle]\!]\,| \\ \prod_{i=1}^{n} l_i[\![p!\langle t\rangle | p!\langle f\rangle]\!]\,|\, l_i[\![d?(x).\mathsf{go}\ k_0.c!\langle x\rangle]\!] \end{array}\right) \not\Downarrow_{\text{SUCC}@l_i}$$

In the above test, we initialise every participant with either t or f, $l_i[\![p!\langle t\rangle|p!\langle f\rangle]\!]$, collect the decision and report it back to the location k_0 on the scoped channel c, $l_i[\![d?(x).\mathsf{go}\ k_0.c!\langle x\rangle]\!]$. Here, a process randomly checks whether two decided values are different. ■

We have just defined termination, validity and agreement for *systems*; in the sequel, we overload these terms - termination, validity and agreement - so that they refer to *configurations*, as long as the network representation of the configuration is *consensus permitting* with respect to the system of the configuration. Finally, we give the definition for an algorithm that solves consensus in typed DπLoc.

Definition 5.2.6 (Consensus in typed DπLoc). A system N solves consensus for n participants if and only iff there exists a consensus permitting network representation Γ for N such that:

1. $\Gamma \triangleright N$ satisfies strong termination
2. $\Gamma \triangleright N$ satisfies validity
3. $\Gamma \triangleright N$ satisfies agreement ■

5.3 Implementing the Consensus-solving Algorithm in typed DπLoc

The DπLoc implementation of the consensus-solving algorithm given by Toueg and Chandra, outlined in § 5.1, is called cons^n and is defined in Table 25. In this table, every participant process is denoted as $\mathsf{P}_i^r\langle v\rangle$, where i is the participant number, r is the current round number and v is the current value estimate held by P_i at round r. $\mathsf{P}_i^r\langle v\rangle$ is defined in terms of three parallel processes:

- $\mathsf{B}_i^r\langle v\rangle$ handles the *broadcasting* of v to all other participants
- $\mathsf{M}_i^r\langle v\rangle$ is a collection of $n-1$ failure detectors (*monitors*), instructing P_i not to wait for value estimates from dead participants

Table 25. *Consensus Algorithm in typed DπLoc*

Broadcasting

$$\mathsf{B}^r_i\langle v\rangle \;\Leftarrow\; \prod_{j=1}^{n} \mathsf{go}\; l_j.s^r_i!\langle v\rangle$$

Failure Detection

$$\mathsf{M}^r_i \;\Leftarrow\; \prod_{i\neq j=1}^{n} \mathsf{monitor}\; l_j\lceil s^r_j!\langle t\rangle\rceil$$

Decision Making

$$\mathsf{R}^r_i\langle t\rangle \;\Leftarrow\; (\nu\, t^r, f^r)\,(\mathsf{S}^r\langle t\rangle \mid \mathsf{T}^r \mid \mathsf{F}^r) \qquad \mathsf{R}^r_i\langle f\rangle \;\Leftarrow\; (\nu\, e^r)\,(\mathsf{S}^r\langle f\rangle \mid \mathsf{C}^r)$$

$$\mathsf{S}^r\langle t\rangle \;\Leftarrow\; \prod_{j=1}^{n} s^r_j?(x).\,\mathsf{if}\;(x=f).\,f^r!\langle\rangle\lceil t^r!\langle\rangle\rceil \qquad \mathsf{S}^r\langle f\rangle \;\Leftarrow\; \prod_{j=1}^{n} s^r_j?(x).\;e^r!\langle\rangle$$

$$\mathsf{F}^r \;\Leftarrow\; f^r?().\mathsf{P}^{r+1}_i\langle f\rangle$$

$$\mathsf{T}^r \;\Leftarrow\; \underbrace{t^r?()...t^r?()}_{n}.\mathsf{P}^{r+1}_i\langle t\rangle \qquad \mathsf{C}^r \;\Leftarrow\; \underbrace{e^r?()...e^r?()}_{n}.\mathsf{P}^{r+1}_i\langle f\rangle$$

Participants

$$\mathsf{P}^r_i\langle v\rangle \;\Leftarrow\; \mathsf{R}^r_i\langle v\rangle \mid \mathsf{B}^r_i\langle v\rangle \mid \mathsf{M}^r_i \quad r<n \qquad \mathsf{P}^{n+1}_i\langle v\rangle \;\Leftarrow\; d!\langle v\rangle$$

Consensus

$$\mathsf{cons}^n \;\Leftarrow\; \prod_{i=1}^{n} l_i[\![p?(x).\mathsf{P}^1_i\langle x\rangle]\!]$$

- $\mathsf{R}^r_i\langle v\rangle$ handles the *receiving* of n value estimates for round r and deciding the value estimate for round $r+1$

For the sake of this case study, requiring us to reason about strong termination, we work at the level of abstraction where the macro monitor $l\lceil P\rceil$ is treated as a basic language construct, with the atomic reduction and transition steps defined in Table 26. This allows us to abstract away from the internal τ-actions associated with repeatedly checking for the liveness of a location k, thereby simplifying our analysis of stabilisation of the algorithm cons^n. We even define this τ-action as a beta move, due to its confluence properties, thereby alleviating much of its burden when considering bisimulations that involve the monitor construct. We leave it up to the interested reader to check that this does not affect the results obtained in § 4.2.3 and § 4.4.

Based on the value set $V = \{t, f\}$, the *decision criteria* is chosen to be *conjunction over booleans*. Using our encoding of booleans in terms of the two free channel names t and f, a branching on the conjunction for two values v and v' can typically be implemented as follows:

- the actual values v and v' can be implemented as outputs on the respective channel. Thus $t!\langle\rangle$ denotes a true whereas $f!\langle\rangle$ denotes a false.
- The branching based on the conjunction for v and v', that is *if* $(v \wedge v')$ *then* P *else* Q

Table 26. *Reduction and Operational Rules for* monitor *in typed DπLoc*

Assuming $\Gamma \vdash l : \textbf{alive}$

(r-mon)
$$\frac{}{\Gamma \triangleright l[\![\text{monitor}\ k\lceil P\rceil]\!] \longrightarrow \Gamma \triangleright l[\![P]\!]}\ \Gamma \nvdash k{:}\textbf{alive}$$

(l-mon)
$$\frac{}{\Gamma \triangleright l[\![\text{monitor}\ k\lceil P\rceil]\!] \xrightarrow{\tau} \Gamma \triangleright l[\![P]\!]}\ \Gamma \nvdash k{:}\textbf{alive}$$

(b-mon)
$$\frac{}{\Gamma \triangleright l[\![\text{monitor}\ k\lceil P\rceil]\!] \overset{\tau}{\longmapsto}_{\beta} \Gamma \triangleright l[\![P]\!]}\ \Gamma \nvdash k{:}\textbf{alive}$$

is then implemented as $t?().t?().P \,|\, f?().Q$. If there are only two boolean values $v!\langle\rangle$, $v'!\langle\rangle$, then we are guaranteed that only one branch, either P or Q will trigger.

The process $\mathsf{R}_i^r\langle v\rangle$ in Table 25 extends this implementation to n boolean values, $v_1 \wedge \ldots \wedge v_n$, for a particular round r. To distinguish these value from boolean values of another round, we use outputs on unique names $t^r!\langle\rangle$ and $f^r!\langle\rangle$. Once again, the branches of the conditional are implemented as two parallel subprocesses:

- F^r is the branch that decides that the estimate for P_i for the next round is false, and according to the decision criteria, it only needs one signalling output on channel f^r to proceed as $\mathsf{P}_i^{r+1}\langle f\rangle$, that is the participant at the next round with an f estimate.
- T^r is the branch that decides that the estimate for next round if true, and needs n signalling outputs on channel t^r to proceed as $\mathsf{P}_i^{r+1}\langle t\rangle$, that is the participant at the next round with an t estimate.

These two branches, $\mathsf{P}_i^{r+1}\langle f\rangle$ and $\mathsf{P}_i^{r+1}\langle t\rangle$, are mutually exclusive *if and only if* the number of boolean values, that is signalling outputs on t^r or f^r, is exactly n. Due to the asynchronous nature of the participants, race conditions may arise that lead to two estimates for a single participant, which would subsequently violate this condition. More specifically, a participant P_j may send its estimate to P_i and then fail, while at the same time, the monitor for P_j at P_i detects the failure and releases the unblocking values estimate for P_j at P_i, thereby resulting in two value estimates for a round r for the same participant P_j at P_i. The implementation in Table 25 solves this problem by packaging estimates from P_j to P_i for round r in channels s_j^r; the process S^r filters multiple estimate values for a single participant P_j to only one estimate, thereby guaranteeing n signalling outputs *only*, on either of the internal channels f^r or t^r.

The implementation of Table 25 also makes an *optimisation* on the algorithm given in [CT96]: from the values set, $V = \{t, f\}$, and the decision criteria, conjunction, we easily conclude that once a participant decides on a false estimate f at round r, it keeps this

estimate for the remaining rounds and finally decides on this value. This happens because it will keep on sending an f estimate to itself for the following rounds, thereby excluding the possibility of receiving n t estimates in any future round $r+m$. As a result, in Table 25, the definition of the $\mathsf{R}_i^r\langle f\rangle$ component in $\mathsf{P}_i^r\langle f\rangle$ does not consider the actual values of the estimates received from other participants. It simply collects n estimates, one for every participant, and progresses to the next round, $r+1$, without performing and decisions or updates to its estimate.

5.4 A Correctness proof for the Consensus Algorithm using Intensional Tests

Our next task is to show that our implementation, cons^n, satisfies the consensus correctness Definition 5.2.6 for DπLoc. We here propose an alternative approach to the proof. According to Definition 5.2.6, we have to show that cons^n stabilises and also that it satisfies weak termination, validity and agreement; the latter three conditions are defined in terms of *four* test contexts that are composed in parallel with cons^n. Instead of proving that cons^n satisfies certain correctness conditions for these four test contexts directly, we propose to construct *three* similar tests for which we have to prove conditions and then show that the necessary correctness conditions required for the four test contexts defined in Definitions 5.2.3, 5.2.4 and 5.2.5 can be extracted from the conditions proved for the three alternative tests.

More specifically, the conditions imposed by Definitions 5.2.3, 5.2.4 and 5.2.5 are at a specification level and their intention is mainly to encode, as close as possible, the traditional definitions given in English; two of these definitions, agreement and validity are defined in terms of negative conditions. *Intensionally* however, these Definitions require that when we initialise every participant in cons^n with either t or f, all participants either decide and agree on a single value or else fail. In addition, they specify an even stricter behaviour for the two extreme cases, that is when the participants are initilialised all with t or all with f: they require that not only participants decide and agree, but they can only agree on the value inputted. Accordingly, we construct three testing systems $\mathsf{req}^n\langle e\rangle$, $\mathsf{req}^n\langle t\rangle$ and $\mathsf{req}^n\langle f\rangle$, defined in Table 27. The first test, $\mathsf{req}^n\langle e\rangle$ encode the intentional behaviour of the generic case while the other two, $\mathsf{req}^n\langle t\rangle$ and $\mathsf{req}^n\langle f\rangle$ encode the intentional behaviour of the two extreme cases just discussed. As before these tests are intended to be composed in parallel with the consensus algorithm cons^n as the system

$$\mathsf{cons}^n \,|\, \mathsf{req}^n\langle v\rangle \quad \text{for } v \in \{t, f, e\}.$$

and produce a signalling output on the fresh channel SUCC at the fresh location k_0 if and only if the intended criteria is met by cons^n.

We claim that our approach to the proof has a number of advantages:

- The first obvious advantage is that we have to prove conditions for *three* tests instead of *four* test. Since, as we shall see, the proofs for the three tests are at most as involving as the required proofs for the four test, the total work required is less.
- Instead of dealing with reasoning about negative barbs, as is required by Definition 5.2.4 and Definition 5.2.5, we can use bisimulation techniques, developed earlier

Table 27. *Requirement Contexts in typed DπLoc*

Initialising

$$\mathsf{I}\langle t\rangle \Leftarrow \prod_{i=1}^{n} l_i[\![p!\langle t\rangle]\!]$$
$$\mathsf{I}\langle f\rangle \Leftarrow \prod_{i=1}^{n} l_i[\![p!\langle f\rangle]\!]$$
$$\mathsf{I}\langle e\rangle \Leftarrow \prod_{i=1}^{n} l_i[\![p!\langle t\rangle \,|\, p!\langle f\rangle]\!]$$

Decision Gathering

$$\mathsf{G} \Leftarrow \prod_{i=1}^{n} l_i[\![d?(x).(d!\langle x\rangle \,|\, \mathsf{go}\; k_0.c_i!\langle x\rangle)]\!] \,|\, k_0[\![\mathsf{monitor}\; l_i\lceil c_i!\langle e\rangle\rceil]\!]$$

Agreement

$$\mathsf{A}^i\langle t\rangle \Leftarrow c_i?(x).\mathsf{if}\; x \in \{t, e\}.\mathsf{A}^{i+1}\langle t\rangle \qquad i \leq n$$
$$\mathsf{A}^i\langle f\rangle \Leftarrow c_i?(x).\mathsf{if}\; x \in \{f, e\}.\mathsf{A}^{i+1}\langle f\rangle \qquad i \leq n$$
$$\mathsf{A}^i\langle e\rangle \Leftarrow c_i?(x).\mathsf{if}\; x = t.\, \mathsf{A}^{i+1}\langle t\rangle\lceil \mathsf{if}\; x = f.\, \mathsf{A}^{i+1}\langle f\rangle\lceil \mathsf{A}^{i+1}\langle e\rangle\rceil\rceil \qquad i \leq n$$
$$\mathsf{A}^{n+1}\langle v\rangle \Leftarrow \textsc{succ}!\langle\rangle \qquad v \in \{t, f, e\}$$

Checking

$$\mathsf{C}\langle v\rangle \Leftarrow (\nu\, c_{1..n})(\mathsf{G} \,|\, k_0[\![\mathsf{A}^1\langle v\rangle]\!]) \qquad v \in \{t, f, e\}$$

Requirement Contexts

$$\mathsf{req}^n\langle v\rangle \Leftarrow \mathsf{I}\langle v\rangle \,|\, \mathsf{C}\langle v\rangle \qquad v \in t, f, e$$

in Chapter 4, to prove conditions about a single positive barb, as is the case for the systems $\mathsf{cons}^n \,|\, \mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$. We claim that with systems the size of cons^n, it can be prohibitive to perform any analysis dealing with negative barbs.

- As we will see in the sequel, the tests $\mathsf{req}^n\langle e\rangle$, $\mathsf{req}^n\langle t\rangle$ and $\mathsf{req}^n\langle f\rangle$, share a lot of common structure, which can be exploited when constructing the witness bisimulations as proofs.
- Most importantly, from our point of view, we get a chance to alleviate some of the burden involved in constructing these witness bisimulations as proofs, by using fault tolerant techniques. We however defer this discussion to § 5.6.

The first of these three tests, $\mathsf{req}^n\langle e\rangle$, checks for the general intentional behaviour: if participants are initialised with a non-homogeneous input, then they either decide and agree on a common value, or else fails. The other two tests, $\mathsf{req}^n\langle t\rangle$ and $\mathsf{req}^n\langle f\rangle$, focus on fringe cases and are more specific regarding the values inputted and decided by the participants: they initialise the participants with a *homogeneous* input (all t and all f respectively) and then check that the decided value is *the same* as the input.

As expected, these three tests are very similar. Each of these tests consists of two sub-systems:

- $\mathsf{I}\langle v\rangle$ initialises the participants with the value v.

- $\mathsf{C}\langle v\rangle$ checks that the decided values are all equal to v.

The latter system, $\mathsf{C}\langle v\rangle$, is decomposed into two further sub-systems:

- G *gathers* the decided values from the participants and returns them to k_0 on the respective scoped channel c_i. It generates *empty* decisions for dead participants.
- $\mathsf{A}^i\langle v\rangle$ checks that all decisions taken by participants $\mathsf{P}_i..\mathsf{P}_n$ reported on the scoped channels $c_i..c_n$ *agree* on v.

In the above definitions, as in the definition of cons^n, the names t and f denote a particular state the participant or test is in; for example, the t in $\mathsf{P}_i^r\langle t\rangle$ denotes that the participant P_i is in a state with estimate t at round r; similarly, the f in $\mathsf{A}^i\langle f\rangle$ denotes that we are currently checking that the participant P_i has decided and agreed on the value f. For this reason, we use an additional value, $v = e$, for the systems of the more generic test, $\mathsf{req}^n\langle e\rangle$, which intuitively stands for *either* value. Thus, $\mathsf{I}\langle e\rangle$ initialises the participants to *either* values t or f and $\mathsf{C}\langle e\rangle$ checks that all the decided values are *either* all t or all f, that is they all agree. Moreover, the process that gathers decided values from participants, generates a value $c_i!\langle e\rangle$, denoting the empty decision, if participant P_i dies; subsequently, the decided value of P_i is *ignored* by the process that checks for agreement, $\mathsf{A}^i\langle v\rangle$. We also note that since gathering is a read operation, after consuming the decide value, we regenerate it again; the fact that we leave this information intact at the participant location happens also to smooth our proofs.

The three agreement processes, $\mathsf{A}^i\langle t\rangle$, $\mathsf{A}^i\langle f\rangle$ and $\mathsf{A}^i\langle e\rangle$, for the tests $\mathsf{req}^n\langle e\rangle$, $\mathsf{req}^n\langle t\rangle$ and $\mathsf{req}^n\langle f\rangle$ deserve further discussion. In particular, we note that $\mathsf{A}^i\langle t\rangle$ and $\mathsf{A}^i\langle f\rangle$ in Table 27 are defined in terms of $\mathsf{A}^{i+1}\langle t\rangle$ and $\mathsf{A}^{i+1}\langle f\rangle$, respectively: this is to say that $\mathsf{A}^i\langle t\rangle$ progresses to $\mathsf{A}^{i+1}\langle t\rangle$ if the value gathered on channel c_i is t; similarly for $\mathsf{A}^i\langle f\rangle$. The other process that checks agreement $\mathsf{A}^i\langle e\rangle$ is defined in terms of $\mathsf{A}^{i+1}\langle t\rangle$, $\mathsf{A}^{i+1}\langle f\rangle$ and $\mathsf{A}^{i+1}\langle e\rangle$. It determines whether the remaining decisions, gathered on $c_i..c_n$, all agree on *either* t or f: if the value on c_i is t, it continues as $\mathsf{A}^{i+1}\langle t\rangle$, which forces the remaining $i+1..n$ decided values to all be t, if the value on c_i is f, it continues as $\mathsf{A}^{i+1}\langle f\rangle$ whereas if the the value on c_i is e, that is empty, it postpones the branching to $\mathsf{A}^r\langle t\rangle$ or $\mathsf{A}^r\langle f\rangle$ till the next round, and thus continues as $\mathsf{A}^{i+1}\langle e\rangle$. The final output on the fresh channel $\textsc{succ}$, indicating that the test was successful, is produced by $\mathsf{A}^{n+1}\langle v\rangle$, where the index $n+1$ means that all the n participants agreed on v.

We conclude this section by showing how the weak termination, validity and agreement properties, defined earlier through four tests in Definition 5.2.3, Definition 5.2.4 and Definition 5.2.5, can be extracted from properties proved about the three tests $\mathsf{req}^n\langle e\rangle$, $\mathsf{req}^n\langle t\rangle$ and $\mathsf{req}^n\langle f\rangle$. Together with the proof stabilisation, we go on to show how to prove that cons^n satisfied consensus.

First we state the property we plan to prove about the three intensional tests $\mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$: we call it *test* (weak) *termination*, which is not to be confused with the consensus weak termination defined earlier.

Definition 5.4.1 (Test Weak Termination). A well-formed configuration $\Gamma \triangleright N$ such that Γ is consensus permitting and $\textsc{succ}, k_0 \notin \mathbf{fn}(N)$ is said to *weakly terminate* up-to m failures

under a test $\mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$, if and only if for all valid dynamic fault contexts F^m where $\Gamma \vdash \mathsf{F}^m$

$$\forall \Gamma', N' \text{ if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle v\rangle \,|\, \mathsf{F}^m \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0}$$

■

Thus we state the property we need to prove for cons^n and the intentional tests.

Proposition 5.4.2 (Test Termination for the Consensus-Solving Algorithm). For any consensus permitting network representation Γ for cons^n, then $\Gamma \triangleright \mathsf{cons}^n$ weakly terminates under any test $\mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$. Stated otherwise:

$$\begin{aligned}
&\forall \Gamma', N' \\
&\quad \text{if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle t\rangle \,|\, \mathsf{F}^m \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0} \\
&\quad \text{if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle f\rangle \,|\, \mathsf{F}^m \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0} \\
&\quad \text{if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle e\rangle \,|\, \mathsf{F}^m \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0}
\end{aligned}$$

We defer the proof of Proposition 5.4.2 to § 5.6, where is it expressed as a fault tolerance problem. For the time being, we will assume it is true and show how the other properties can be extracted from this result.

We next prove the following two lemmas, the premises of which are the *negated* validity and agreement conditions.

Lemma 5.4.3 (Extracting Validity). Consider a consensus-permitting network Γ, such that $\Gamma \triangleright \mathsf{cons}^n$ is well-formed. For any dynamic fault context F^{n-1} such that $\Gamma \vdash \mathsf{F}^{n-1}$

- If $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, \prod_{i=1}^{n} l_i[\![p!\langle t\rangle]\!] \,|\, l_i[\![d?(x).\mathsf{if}\ (x = f).\mathsf{go}\ k_0.\textsc{succ}!\langle\rangle]\!] \Downarrow_{\textsc{succ}@k_0}$ then $\exists \Gamma', N'$ such that $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, \mathsf{req}^n\langle t\rangle \longrightarrow^\star \Gamma' \triangleright N' \not\Downarrow_{\textsc{succ}@k_0}$
- If $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, \prod_{i=1}^{n} l_i[\![p!\langle f\rangle]\!] \,|\, l_i[\![d?(x).\mathsf{if}\ (x = t).\mathsf{go}\ k_0.\textsc{succ}!\langle\rangle]\!] \Downarrow_{\textsc{succ}@k_0}$ then $\exists \Gamma', N'$ such that $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, \mathsf{req}^n\langle f\rangle \longrightarrow^\star \Gamma' \triangleright N' \not\Downarrow_{\textsc{succ}@k_0}$

Proof. We here prove the first clause, and leave the second similar clause for the interested reader. Assume

$$\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, \prod_{i=1}^{n} l_i[\![p!\langle t\rangle]\!] \,|\, l_i[\![d?(x).\mathsf{if}\ (x = f).\mathsf{go}\ k_0.\textsc{succ}!\langle\rangle]\!] \Downarrow_{\textsc{succ}@k_0} \tag{5.1}$$

Since

$$\textsc{succ} \notin \mathbf{fn}(\mathsf{cons}^n, \mathsf{F}^{n-1}, \prod_{i=1}^{n} l_i[\![p!\langle t\rangle]\!])$$

Then this barb can only be caused by the subsystem $\prod_{i=1}^{n} l_i[\![d?(x).\mathsf{if}\ (x = f).\mathsf{go}\ k_0.\textsc{succ}!\langle\rangle]\!]$. However,

$$\prod_{i=1}^{n} \Gamma \triangleright l_i[\![d?(x).\mathsf{if}\ (x = f).\mathsf{go}\ k_0.\textsc{succ}!\langle\rangle]\!] \not\longrightarrow$$

and from the structure of $\prod_{i=1}^{n} l_i[\![d?(x).\mathsf{if}\ (x = f).\mathsf{go}\ k_0.\textsc{succ}!\langle\rangle]\!]$ we conclude that the only way we can have (5.1) is, if

$$\exists\, \Gamma', N', l_i \text{ for } 1 \leq i \leq n \text{ such that } \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, \prod_{i=1}^{n} l_i[\![p!\langle t\rangle]\!] \longrightarrow^{\star} \equiv \Gamma' \triangleright N' \,|\, l_i[\![d!\langle f\rangle]\!] \tag{5.2}$$

This is the required condition needed to prove the implication. By (5.2) we conclude that the gathering sub-system G in $\mathsf{req}^n\langle t\rangle$ can produce $c_i!\langle f\rangle$ at k_i. Thus, even if for the remaining $1 \leq j \leq n$ where $j \neq i$ the G sub-system can produce $c_j!\langle v_j\rangle$ at k_0 where v_j are either t or e, $\mathsf{C}\langle t\rangle$ may still block because when we reach $\mathsf{A}^i\langle t\rangle$, the input defined as $c_i?(x).\mathsf{if}\ x \in \{t, e\}.\mathsf{A}^{i+1}\langle t\rangle$ may consume $c_i!\langle f\rangle$ which will not branch to $\mathsf{A}^{i+1}\langle t\rangle$ since $f \notin \{t, e\}$. □

Lemma 5.4.4 (Extracting Agreement). Consider a consensus-permitting network Γ, such that $\Gamma \triangleright \mathsf{cons}^n$ is well-formed. Then for any dynamic fault context F^{n-1} such that $\Gamma \vdash \mathsf{F}^{n-1}$ if

$$\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, (\nu\, c) \left(\begin{array}{l} k_0[\![c?(x).c?(y).\mathsf{if}\ (x \neq y).\textsc{succ}!\langle\rangle]\!] \,| \\ \prod_{i=1}^{n} l_i[\![p!\langle t\rangle]\!] \,|\, l_i[\![p!\langle f\rangle]\!] \,|\, l_i[\![d?(x).\mathsf{go}\ k_0.c!\langle x\rangle]\!] \end{array} \right) \Downarrow_{\textsc{succ}@k_0}$$

then

$$\exists\, \Gamma', N' \text{ such that } \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, \mathsf{req}^n\langle e\rangle \longrightarrow^{\star} \Gamma' \triangleright N' \not\Downarrow_{\textsc{succ}@k_0}$$

Proof. The proof for this lemma is similar to that of Lemma 5.4.3. By a similar analysis we conclude that the gathering sub-system G in $\mathsf{req}^n\langle e\rangle$ can produce $c_i!\langle f\rangle$ at $c_j!\langle t\rangle$ for $1 \leq i, j \leq n$. We thus conclude that $\mathsf{C}\langle e\rangle$ may block. Assuming that $i < j$, then we have the following cases

- If we have $\mathsf{A}^i\langle t\rangle$ we trivially block.
- If we have $\mathsf{A}^i\langle e\rangle$ then by consuming $c_i!\langle f\rangle$ first (since $i < j$) we proceed as $\mathsf{A}^i + 1\langle f\rangle$; if this reaches $\mathsf{A}^j\langle f\rangle$ it may block by consuming $c_j!\langle t\rangle$
- We have a similar case if we have $\mathsf{A}^i\langle f\rangle$. □

From Proposition 5.4.2, we can conclude that the premises of Lemma 5.4.3 and Lemma 5.4.4 are always false, and by double negation we obtain validity and agreement for cons^n. We next prove a more straightforward proof for extracting consensus weak termination.

Lemma 5.4.5 (Extracting Weak Termination). If $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle$ satisfies *test* weak termination (Definition 5.4.1), then $\Gamma \triangleright \mathsf{cons}^n$ satisfies *consensus* weak termination (Definition 5.2.3).

Proof. If we expand Definition 5.4.1, that is test weak termination, and the Definition of $\mathsf{req}^n\langle e\rangle$ from Table 27, the by our assumption we have:

$$\forall \Gamma', N' \text{ and } \mathsf{F}^{n-1} \text{ such that } \Gamma \vdash \mathsf{F}^{n-1}$$
$$\text{if } \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, (\nu\, c_{1..n}) \left(\begin{array}{l} k_0[\![\mathsf{A}^1\langle e\rangle]\!] \,| \\ \prod_{i=1}^{n} \left(\begin{array}{l} l_i[\![p!\langle t\rangle \,|\, p!\langle f\rangle]\!] \,| \\ l_i[\![d?(x).\mathsf{go}\; k_0.c_i!\langle x\rangle]\!] \,| \\ k_0[\![\mathsf{monitor}\; l_i\lceil c_i!\langle e\rangle\rceil]\!] \end{array} \right) \end{array} \right) \longrightarrow^* \Gamma' \triangleright N'$$
$$\text{then } \Gamma' \triangleright N' \Downarrow_{\text{succ}@k_0}$$

By expanding Definition 5.2.3, that is consensus weak termination, then we require:

$$\forall \Gamma', N' \text{ and } \mathsf{F}^{n-1} \text{ such that } \Gamma \vdash \mathsf{F}^{n-1}$$
$$\text{if } \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{F}^{n-1} \,|\, (\nu\, c_{1..n}) \left(\begin{array}{l} k_0[\![c_1?()...c_n?().\text{succ}!\langle\rangle]\!] \,| \\ \prod_{i=1}^{n} \left(\begin{array}{l} l_i[\![p!\langle t\rangle \,|\, p!\langle f\rangle]\!] \,| \\ l_i[\![d?(x).\mathsf{go}\; k_0.c_i!\langle\rangle]\!] \,| \\ k_0[\![\mathsf{monitor}\; l_i\lceil c_i!\langle\rangle\rceil]\!] \end{array} \right) \end{array} \right) \longrightarrow^* \Gamma' \triangleright N'$$
$$\text{then } \Gamma' \triangleright N' \Downarrow_{\text{succ}@k_0}$$

But this is close to our assumption with the exception of the agreement function. More specifically, in consensus weak termination we have $k_0[\![c_1?()...c_n?().\text{succ}!\langle\rangle]\!]$ instead of $k_0[\![\mathsf{A}^1\langle e\rangle]\!]$, that is just collecting instead of collecting and matching the decided values. By virtue of the fact that the implication is a simplification of the assumption, the lemma is trivially true. ◻

Once again, from Proposition 5.4.2, we immediately obtain weak termination for cons^n as a result of Lemma 5.4.5. The only remaining result that needs to be proved about cons^n is stabilisation, which we now state and prove.

Lemma 5.4.6 (Consensus-Solving Algorithm Stabilisation). For a suitable consensus-permitting network representation Γ, the configuration $\Gamma \triangleright \mathsf{cons}^n$ stabilises. More formally

$$\exists\, n \text{ such that } \forall \Gamma', N' \text{ if } \Gamma \triangleright \mathsf{cons}^n \underbrace{\longrightarrow \cdot \ldots \cdot \longrightarrow}_{\leq n} \Gamma' \triangleright N' \text{ then } \Gamma' \triangleright N' \not\longrightarrow$$

Proof. A static analysis of the cons^n implementation of Table 25 reveals that no replicated input is used. Since this is the only way infinite computation can be achieved in typed DπLoc, we immediately conclude that there exists an upper limit on the number of reductions possible. ◻

We conclude the section by giving the final theorem, showing that our implemenatation, cons^n, satisfies consensus.

Theorem 5.4.7 (Consensus Satisfaction). The implementation cons^n of Table 25 satisfies consensus.

Proof. We recall that for cons^n to satisfy consensus, for a suitable consensus-permitting network representation Γ we have to show that:

1. $\Gamma \triangleright \mathsf{cons}^n$ stabilises.
2. $\Gamma \triangleright \mathsf{cons}^n$ satisfies *weak* termination
3. $\Gamma \triangleright \mathsf{cons}^n$ satisfies validity
4. $\Gamma \triangleright \mathsf{cons}^n$ satisfies agreement

By Lemma 5.4.6 we have condition 1, stabilisation.

From Proposition 5.4.2, that is weak test termination of cons^n, and Lemma 5.4.5, extracting weak termination, we obtain condition 2, weak termination and thus together with condition 1 we have strong termination.

From Proposition 5.4.2, that is weak test termination of cons^n, and Lemma 5.4.3, extracting validity, we obtain condition 3, validity.

Finally, from Proposition 5.4.2, that is weak test termination of cons^n, and Lemma 5.4.4, extracting agreement, we obtain condition 4, agreement.

Thus by Definition 5.2.6, cons^n satisfies consensus. □

With this result, we know that the proof that cons^n satisfies consensus hinges on proving that cons^n satisfies test weak termination for the tests $\mathsf{req}^n\langle t\rangle$, $\mathsf{req}^n\langle f\rangle$ and $\mathsf{req}^n\langle e\rangle$, that is Proposition 5.4.2. The remainder of the Chapter is dedicated for this proof, which we express as a fault tolerant problem and prove using the bisimulation techniques developed in Chapter 4.

5.5 Analysis of the Consensus-solving Algorithm and Tests

Before we delve into the actual proofs, we analyse the dynamic characteristics of the systems at hand and develop notation that enables us to describe the significant states of their execution. This notation will then facilitate the presentation of our proofs in terms of witness bisimulations.

We note that the implementation of the participants making up the algorithm cons^n of Table 25 consists entirely of asynchronous outputs that migrate between locations and are consumed by a limited number of inputs that in turn generate more asynchronous outputs. More specifically, the asynchronous outputs of a participant go through successive iterations of migration, filtering and deciding, making up a round.

Migration The broadcasting and monitor sub-systems, defined as $\mathsf{B}^r_i\langle v\rangle$ and $\mathsf{M}^r_j\langle V\rangle$ in Table 25, are the only entities affecting the migration of an asynchronous message $s^r_i!\langle v\rangle$ from l_i to l_j. Thus we generally have the following setting for a particular round r

$$l_i[\![\mathsf{go}\ l_j.s^r_i!\langle v\rangle]\!] \,|\, l_j[\![\mathsf{monitor}\ l_i\lceil s^r_i!\langle t\rangle\rceil]\!] \,|\, M$$

where $l_i[\![\mathsf{go}\ l_j.s^r_i!\langle v\rangle]\!]$ tries to send $s^r_i!\langle v\rangle$ to l_j, at the destination $l_j[\![\mathsf{monitor}\ l_i\lceil s^r_i!\langle t\rangle\rceil]\!]$ is ready to launch $s^r_i!\langle t\rangle$ at l_j if l_i fails, and M cannot produce any more outputs on the channel s^r_i at location l_j. There are three sub-cases for migration we need to consider in our bisimulations, depending on the liveness of l_i and l_j:

- If the destination, l_j, is dead, then we have a β-move for $l_i[\![\mathsf{go}\ l_j.s^r_i!\langle v\rangle]\!]$ using (b-ngo) defined in Table 22.

- If the source destination, l_i, is dead, then we also have a β-move, this time for $l_j[\![\mathsf{monitor}\ l_i\lceil s_i^r!\langle t\rangle\rceil]\!]$ using (b-mon) of Table 26.
- If both the source and destination locations are alive, then we have a scenario where $\mathsf{go}\,l_j.s_i^r!\langle v\rangle$ may reach its destination, but also, if l_i subsequently fails, the possibility of $\mathsf{monitor}\ l_i\lceil s_i^r!\langle t\rangle\rceil$ releasing its message at l_j. This leads to a race condition where we have two estimate for P_i at P_j, which may in turn affect our implementation of the conditional over booleans, which requires exactly n values to function properly. This however, turns out not to be important for our analysis, as we discuss in the next step.

Filtering To avoid the problem of having two round values for a particular participant we use a filtering mechanism defined by the process S^r in Table 25. Apart from the β-moves generated by name matching on the values inputted, we may have two possible cases for a filtering input $s_i^r?(x).\,\mathsf{if}\ (x = f).\ f^r!\langle\rangle\lceil\, t^r!\langle\rangle\rceil$

- The first case is when we have a single output for the filtering input,that is

$$l_j[\![s_i^r!\langle v\rangle]\!] \mid l_j[\![s_i^r?(x).\,\mathsf{if}\ (x = f).\ f^r!\langle\rangle\lceil\, t^r!\langle\rangle\rceil]\!]$$

- The second case is when we have a two outputs for the filtering input, as outlined above

$$l_j[\![s_i^r!\langle v\rangle]\!] \mid l_j[\![s_i^r!\langle t\rangle]\!] \mid l_j[\![s_i^r?(x).\,\mathsf{if}\ (x = f).\ f^r!\langle\rangle\lceil\, t^r!\langle\rangle\rceil]\!]$$

It turns out that it is not important how many outputs on channel s_i^r there are as long as there is at least one; what affects the execution is the value of the asynchronous output selected, since this affects the decision process.

Deciding The decision processes is made up of n values for round r, generated by the filtering stage, together with the sub-processes T^r and F^r defined in Table 25. Once the last input of either branch T^r and F^r is consumed, we start all over again for the following round $r+1$ or else decide. Here we also have two possible cases:

- In the first case we only have filtered t values for round r so far, that is

$$\underbrace{t^r!\langle\rangle|\ldots|t^r!\langle\rangle}_{\leq n} \mid f^r?().\mathsf{P}_i^{r+1}\langle f\rangle \mid \underbrace{t^r?()...t^r?()}_{\leq n}.\mathsf{P}_i^{r+1}\langle t\rangle$$

At this point, we can still take either branches of the conditional, proceeding as $\mathsf{P}_i^{r+1}\langle t\rangle$ or $\mathsf{P}_i^{r+1}\langle f\rangle$.

- In the second case we have filtered at least one f values for round r, that is

$$\underbrace{f^r!\langle\rangle|\ldots|f^r!\langle\rangle}_{\geq 1} \mid \underbrace{t^r!\langle\rangle|\ldots|t^r!\langle\rangle}_{\leq n} \mid f^r?().\mathsf{P}_i^{r+1}\langle f\rangle \mid \underbrace{t^r?()...t^r?()}_{\leq n}.\mathsf{P}_i^{r+1}\langle t\rangle$$

At this point, we know that we cannot generate the necessary n t values to proceed as $\mathsf{P}_i^{r+1}\langle t\rangle$. Thus only the f branch can be taken, proceeding as $\mathsf{P}_i^{r+1}\langle f\rangle$.

Because of our optimisation in the algorithm, when a participant has an f estimate at round r, deciding translates to merely collecting the estimates, with only one possible branch for the next round. We therefore have the case where:

$$\underbrace{c^r!\langle\rangle|\ldots|c^r!\langle\rangle}_{\leq n} \mid \underbrace{c^r?()...c^r?()}_{\leq n}.\mathsf{P}_i^{r+1}\langle f\rangle$$

Apart from migrating, filtering and deciding, and β-moves such as process spawning using (l-fork), every participant produces also junk moves, where asynchronous outputs are generated but there are no corresponding inputs to consume them; examples are monitors that trigger and produce outputs for past rounds.These moves are not significant for the outcome of the decided value of the participant.

A similarly analysis can be carried out for the tests, $\mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$ that are also based on asynchronous outputs. The core activity in these tests is carried out by the sub-systems $\mathsf{C}\langle v\rangle$.

- The gathering sub-system, G, yields cases very similar to those considered above in asynchronous output migration for participants; the only difference is that there is one less possible beta move since the destination, k_0, never fails.
- The agreement sub-system, $\mathsf{A}^i\langle v\rangle$ is very similar to the filtering stage discussed above, where it only considers one collected value from the gathering sub-system; just like in the case of participants, the gathering system may collect two values for a particular participant, the actual value decided and an empty value in case the participant dies after it decides. In addition, the sub-system $\mathsf{A}^i\langle v\rangle$ performs names matching comparisons on the values collected, which are β-moves.

It turns out that only a subset of the actions mentioned above are significant for our bisimulations. More specifically, a large number of these τ-actions do not affect the state of the participant as it progresses from one round to another. By using bisimulations and fault-tolerant simulations up-to β-moves, we automatically weed out all the unnecessary β-moves. This still leaves us a considerable number or actions that yield confluent intermediary states for a participant; typical examples are junk moves or communication where there can only be one input and one output.

One possible approach to deal with these τ-moves would be to develop further β-moves; these however, turn out to be less straight forward compared to the β-moves defined so far, because we need to defined certain conditions for code executing in parallel to the located processes causing the β-move. In view that up-to techniques are delicate to set up and the fact that the constraints of these latter moves may be difficult to specify, we forgo this approach.

Instead we develop notation for participant *aggregate states*, which ranges over a number of intermediary states a participant may be in as it progresses from one round to the other. In essence, aggregate states represent code at location l_i which satisfies certain conditions. This level of abstraction proves to be adequate for the presentation of the subsequent bisimulations.

Notation 5.5.1. We find it convenient to use the shorthand notation N_i, for a system N located entirely at l_i, that is

$$N_i \stackrel{\text{def}}{=} N \equiv (\nu\,\tilde{n})l_i[\![P]\!] \,|\, \ldots \,|\, l_i[\![Q]\!]$$

We also use the shorthand N_0 for a group of processes that is entirely located at k_0 ■

Participant aggregate states are defined in terms of two variables, that is r the current round number and $v \in \{t, f\}$, the current estimate at round r. Thus $\mathsf{Pnt}_i(r, v)$ ranges over all the code residing at l_i, that N_i, comprising of

- all the code generated by $\mathsf{P}_i^r\langle v\rangle$.
- residual code from previous rounds $r - m$.
- all the asynchronous messages received at l_i from other participants.
- code placed at l_i by the test contexts $\mathsf{req}^n\langle v\rangle$.

Due to our particular implementation optimisations, $\mathsf{Pnt}_i(r, t)$, the aggregate state for a participant with a t estimate and its dual, $\mathsf{Pnt}_i(r, f)$, the aggregate state for a participant with an f estimate, have a different structure and require distinct definitions.

Definition 5.5.2 (Participant Aggregate State with a *t* estimate). $\mathsf{Pnt}_i(r, t)$ ranges over all the intermediary states of a participant P_i at round r with estimate t and is defined as:

$$\mathsf{Pnt}_i(r,t) \stackrel{\text{def}}{=} \begin{cases} (\nu\, t^r, f^r, \tilde{n})\ l_i[\![f^r?().P]\!] \,|\, l_i[\![t^r?().Q]\!] \,|\, N_i \quad \text{where } N_i \not\equiv N_i' \,|\, l_i[\![f^r!\langle\rangle]\!],\ 1 \leq r \leq n \\ (\nu\tilde{n})(l_i[\![p?(x).\mathsf{P}_i^r\langle x\rangle]\!]) \,|\, l_i[\![p!\langle t\rangle]\!] \,|\, N_i \quad \text{where } N_i \not\equiv N_i' \,|\, l_i[\![p!\langle f\rangle]\!] \\ l_i[\![d!\langle t\rangle]\!] \,|\, N_i \end{cases}$$

Intuitively, $\mathsf{Pnt}_i(r, t)$ refers to all the code at l_i while the participant is *still* deciding which estimate to adopt for round $r + 1$, which is why both the f branch, $l_i[\![f^r?().P]\!]$, and the t branch, $l_i[\![t^r?().Q]\!]$, are still not taken. The conditions imposed on $\mathsf{Pnt}_i(r, t)$ make it such that it could not refer to $\mathsf{Pnt}_i(r-m, t)$ at some earlier round $r-m$, because for those rounds, only one branch, the f branch, would be available; it could neither refer to a future round $\mathsf{Pnt}_i(r + m, t)$ either, because these branches would not have been launched yet.

The condition imposed on the remaining code at l_i, $N_i \not\equiv N_i' \,|\, l_i[\![f^r!\langle\rangle]\!]$, refers to the second case of the decision phase discusses earlier in this section: If $N_i \equiv N_i' \,|\, l_i[\![f^r!\langle\rangle]\!]$ then, even though both branches are still available, only the f branch can be taken since n messages of the form $l_i[\![t^r!\langle\rangle]\!]$, required by the t branch to proceed, can never be generated.

$\mathsf{Pnt}_i(r, t)$ also ranges over two fringe cases: when $r = 1$ the notation ranges over the case when a participant is initialised with a t value only; When $r = n + 1$ it ranges over the case when it has decided the value t. ■

From all the τ-actions discussed above, the crucial action that denotes a change in the state of a participant in Definition 5.5.2, is that is that of the decision, more specifically, when either the t branch or the f branch is taken. The definition of the aggregate state of a participant with an f estimate is similar.

Definition 5.5.3 (Participant Aggregate State with a *f* estimate). $\mathsf{Pnt}_i(r, f)$ ranges over all the intermediary states of a participant P_i at round r with estimate f and is defined as:

$$\mathsf{Pnt}_i(r,f) \stackrel{\text{def}}{=} \begin{cases} (\nu\, f^{r-1}, \tilde{n})\ l_i[\![f^{r-1}!\langle\rangle]\!] \,|\, l_i[\![f^{r-1}?().P]\!] \,|\, N_i \quad \text{for } 1 \leq r \leq n \\ (\nu\, e^r, \tilde{n})\ l_i[\![e^r?().P]\!] \,|\, N_i \quad \text{for} \quad 1 \leq r \leq n \\ (\nu\tilde{n})(l_i[\![p?(x).\mathsf{P}_i^r\langle x\rangle]\!]) \,|\, l_i[\![p!\langle f\rangle]\!] \,|\, N_i \quad \text{where } N_i \not\equiv N_i' \,|\, l_i[\![p!\langle t\rangle]\!] \\ l_i[\![d!\langle f\rangle]\!] \,|\, N_i \end{cases}$$

First of all, $\mathsf{Pnt}_i(r, f)$ ranges over participants that are still in the previous round $r-1$ with the currently held estimate being t, and at least one f estimate filtered for that round. As we already discussed, the only branch that can be taken in this case is the f branch proceeding as $\mathsf{P}_i^r\langle f\rangle$.

Similar to the previous aggregate state with a t estimate, $\mathsf{Pnt}_i(r, f)$ refers to all the code at l_i while the participant is still collecting estimates for round r, and is thus identified by the input on channel e^r, $l_i[\![e^r?().P]\!]$.

$\mathsf{Pnt}_i(r, f)$ also ranges over two fringe states: the case where $r = 1$ is similar to Definition 5.5.2, whereas the case for $r = n+1$ states that the participant must have decided the value f *and* completed all of its estimate broadcasts. ■

There is only one participant aggregate state left and we define it next.

Definition 5.5.4 (Uninitialised Participant Aggregate State). $\mathsf{Pnt}_i(0)$ ranges over all the states of a participant P_i before round 1, when it is initialised with either t or f, defined as:

$$\mathsf{Pnt}_i(0) \stackrel{\text{def}}{=} (\nu\,\tilde{n})(N_i \,|\, l_i[\![p?(x).\mathsf{P}_i^r\langle x\rangle]\!]) \,|\, l_i[\![p!\langle t\rangle]\!] \,|\, l_i[\![p!\langle f\rangle]\!]$$

■

Finally, we define aggregate states for the tests located at k_0.

Definition 5.5.5 (Test Aggregate State). We define two aggregate states for tests, Test_i and $\mathsf{Test}_{\text{done}}$, ranging over code located at k_0 and defined as

$$\mathsf{Test}_i \stackrel{\text{def}}{=} \begin{cases} k_0[\![c_i?(x).P]\!] \,|\, N_0 & \text{for } 1 \le i \le n \\ k_0[\![\textsc{succ}!\langle\rangle]\!] \,|\, N_0 & \text{for } i = n+1 \end{cases}$$

$$\mathsf{Test}_{\text{done}} \stackrel{\text{def}}{=} N_0 \quad \text{such that } N_0 \not\equiv N_0' \,|\, k_0[\![c_i?(x).P]\!] \text{ and } N_0 \not\equiv N_0' \,|\, k_0[\![\textsc{succ}!\langle\rangle]\!]$$

For $1 \le i \le n$, Test_i is defined in terms of how many decided values they still need to check. We define a fringe states for when all the checks are successful, Test_{n+1}. We define an additional aggregate state, $\mathsf{Test}_{\text{done}}$, for when the signalling fresh output $k_0[\![\textsc{succ}!\langle\rangle]\!]$ is consumed. ■

The witness bisimulations and fault tolerant simulations we given in the next chapter are presented in terms of the aggregate states defined in Definition 5.5.2, 5.5.3, 5.5.4 and 5.5.5. From these definitions, we can easily conclude that a transition from one aggregate state to another happens only as a result of communications between asynchronous outputs and inputs. For instance, a participant $\mathsf{Pnt}_i(r, t)$ transitions to $\mathsf{Pnt}_i(r+1, t)$ when the last input on $t^r?().P$ is triggered by the nth asynchronous output $t^r!\langle\rangle$, releasing P which contains the branches T^{r+1} and F^{r+1}, as required by $\mathsf{Pnt}_i(r+1, t)$; similarly $\mathsf{Pnt}_i(r, t)$ transitions to $\mathsf{Pnt}_i(r+1, f)$ when one of the filtering processes $s_j^r?(x).\mathsf{if}\, x = f.f^r!\langle\rangle\lceil t^r!\langle\rangle\rceil$ reacts with an asynchronous output $!\langle s_j^r\rangle f$ and produces $f^r!\langle\rangle$.

Since neither of the β-moves defined in Table 22 and Table 26, lead to any transition in aggregate states, the witness bisimulations and fault tolerant simulations abstract over β-moves and are thus bisimulations and fault tolerant simulations up-to β-moves. These bisimulations and fault tolerant simulations also abstract over other confluent τ-moves

when using aggregate states, such as the junk moves mentioned earlier, thereby making our presentation more understandable and manageable.

5.6 Intentional Test Termination for the Consensus-Solving Algorithm

This final Section deals exclusively with the proof for Proposition 5.4.2, stated earlier in § 5.4, asserting that the consensus-solving algorithm, cons^n satisfies test termination for the intentional tests $\mathsf{req}^n\langle t\rangle$, $\mathsf{req}^n\langle f\rangle$ and $\mathsf{req}^n\langle e\rangle$. As a result of this proof, we also prove that cons^n satisfies consensus, as described earlier in § 5.4.

Proposition 5.4.2 requires that we consider the behaviour of the systems $\mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle$, $\mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle$ and $\mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle$ under *any* valid fault-contexts F^{n-1} and prove that certain conditions are satisfied despite of the faults induced; see Definition 5.4.1 for test termination. Even though the number of valid fault contexts is bounded by n, such an analysis can be quite laborious and tedious since it involves a lot of repeated work in each case. To alleviate this burden, our proof of Proposition 5.4.2 is split into two parts:

- In the first part, we consider the behaviour of $\mathsf{cons}^n \,|\, \mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$ in *failure-free* setting and ensure that it does weakly terminate. It turns out that for this proof, we only require three failure-free witness bisimulations up-to β-moves.
- The analysis of the behaviour of $\mathsf{cons}^n \,|\, \mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$ is relegated to the second part, where we prove, that the behaviour of $\mathsf{cons}^n \,|\, \mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$ in *failure-free* setting is *preserved* under any F^{n-1} fault context. According to Proposition 4.4.3 and Proposition 4.4.11, for this proof we only require three witness $(n-1)$-fault-tolerant simulations up-to β-moves.

Merging these two proofs, we obtain the required property that test termination is attained by cons^n, for any test $\mathsf{req}^n\langle t\rangle$, $\mathsf{req}^n\langle f\rangle$ and $\mathsf{req}^n\langle e\rangle$, under any valid fault-contexts F^{n-1}.

Lemma 5.6.1 (Test Termination in a Failure-Free Setting). For any valid consensus-permitting network Γ such that $\Gamma \triangleright \mathsf{cons}^n$ is well formed, then $\Gamma \triangleright \mathsf{cons}^n$ weakly terminates in a failure-free setting under any test $\mathsf{req}^n\langle t\rangle$,$\mathsf{req}^n\langle f\rangle$ and $\mathsf{req}^n\langle e\rangle$.

Proof. According to Definition 5.4.1, we have to show that

$$\forall \Gamma', N' \text{ if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle v\rangle \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0}$$

for $v \in \{t, f, e\}$. It turns out that $\textsc{succ}@k_0$ is the only barb that can be produced by $\Gamma \triangleright N \,|\, \mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$. Moreover, a configuration that trivially satisfies the condition

$$\forall N' \text{ such that } \Gamma \triangleright N \longrightarrow^* \Gamma \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0}$$

and also has $\textsc{succ}@k_0$ as the only barb is $\Gamma \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!]$. Thus, the definition of failure-free reduction barb congruence, developed in Chapter 4, which guarantees reduction closure and barb preservation, can be used to prove Lemma 5.6.1. In other words, Lemma 5.6.1 can be proved by showing the following three clauses:

1. $\Gamma \models k_0[\![\textsc{succ}!\langle\rangle]\!] \;\cong_{ff}\; \mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle$

2. $\Gamma \models k_0[\![\text{succ}!\langle\rangle]\!] \;\; \cong_{ff} \;\; \mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle$

3. $\Gamma \models k_0[\![\text{succ}!\langle\rangle]\!] \;\; \cong_{ff} \;\; \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle$

To prove these three statements, we use the failure-free bisimulation relation up-to-β-moves, developed in Chapter 4. We also note that since we consider a failure-free setting, the number of transitions we need to consider in these witness bisimulations is largely minimised since locations never fail. We here mention two example transitions that are simplified in a failure free setting:

- We are guaranteed that every migration will succeed. An further optimisation that could be adopted for this τ-action would be to make (l-go) a β-move, so that we can completely abstract from it. Nevertheless with our use of aggregate states, our presentation is not affected.
- The failure detectors used by the participants and tests never trigger: thus we never have race conditions with outputs (one from the participant and one from the failure detector) for a single input. This one-to-one relationship between every output and input minimises the range of states that need to be considered.

The proof for clause 1 is given by the failure-free bisimulation relation up-to-β-moves, $\mathcal{R}_e$, defined below.

$$\mathcal{R}_e \stackrel{\text{def}}{=} \left\{ \begin{array}{lll}
\Gamma \triangleright k_0[\![\text{succ}!\langle\rangle]\!] & , \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle & \\
\Gamma \triangleright k_0[\![\text{succ}!\langle\rangle]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_j \,|\, \prod_{i=1}^{n} N_i) & \left| \begin{array}{l} 1 \le j \le n+1 \\ N_i = \mathsf{Pnt}_i(0) \text{ or } \mathsf{Pnt}_i(1,t) \\ \qquad \text{or } \mathsf{Pnt}_i(1,f) \end{array} \right. \\
\Gamma \triangleright k_0[\![\text{succ}!\langle\rangle]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_j \,|\, \prod_{i=1}^{n} \mathsf{Pnt}_i(r_i,t)) & \left| \begin{array}{l} 1 \le j \le n+1 \\ 1 \le r_i \le n+1 \end{array} \right. \\
\Gamma \triangleright k_0[\![\text{succ}!\langle\rangle]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_j \,|\, \prod_{i=1}^{n} N_i) & \left| \begin{array}{l} 1 \le j \le n+1 \\ N_i = \mathsf{Pnt}_i(1,t) \text{or } \mathsf{Pnt}_i(r_i,f) \\ \;\text{for } 1 \le r_i \le n+1 \end{array} \right. \\
\Gamma \triangleright k_0[\![\mathbf{0}]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_{\text{done}} \,|\, \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1,t)) & \\
\Gamma \triangleright k_0[\![\mathbf{0}]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_{\text{done}} \,|\, \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1,f)) &
\end{array} \right\}$$

$\mathcal{R}_e$ captures the fact that no matter what value the participants are initialised to, they will all reach agreement by the second round:

- If the participants happen to be all initialised to t in round 1, that is they start in agreement, then they will all broadcast a t value, Since these broadcasts are never interfered by failure, they will all complete successfully and every participant will receive n t estimates for round 1. Hence the participants all progress to round 2 using the t branch.

- If at least one of the participants is initialised with f at round 1, then no participant can progress to round 2 with a t estimate because they can at most receive $n-1$ t estimates. As a result, the only branch that can be taken by every participant is the f branch, using the f estimate broadcasted by the participant initialised with f at round 1, and by round 2, all participants will agree on f.

Once agreement is attained in round 2, it is maintained throughout the remaining rounds: if the agreement at round 2 is on t, then we apply the same reasoning used for the the first case above; if the agreement at round 2 is on f, then from the implementation optimisation of Table 25, it is easy to see that no decision (or update) is used and that this estimate is immutable for the remaining rounds.

If we had to explain all this in terms of the possible transitions in $\mathcal{R}_e$, as participants start to get initialised with either t or f values, we transition from the first clause to the second clause in $\mathcal{R}_e$. If we reach a state where all participants are initialised with a t value, we transition to the third clause of $\mathcal{R}_e$ where participants can move to rounds $2..n+1$ with a t estimate; we note that participants move in *lockstep*, meaning that their round number never differs by more that 1. If, on the other hand, we are in the second clause and reach a state where all participants are initialised and at least one participant is initialised with an f estimate, we transition to the fourth clause where in order to progress to rounds $2..n+1$ every participant must have an f estimate. As participants reach round $n+1$ in the third and fourth clauses, Test_1 can transition to Test_{n+1} while still remaining at the the third and fourth clauses respectively. If the left hand side produces the action $k_0 : \textsc{succ}!\langle\rangle$, then the right hand side produces a matching weak action transitioning from the third and fourth clause to the fifth and six clauses respectively in $\mathcal{R}_e$.

The bisimulation relations up-to-β-moves $\mathcal{R}_t$ and $\mathcal{R}_f$ are the witness relations proving clauses 2 and 3, that is when participants are either all initialised with a t value or else all initialised with an f value. We leave the reader to check all the possible transitions; they are similar to the transitions discussed for $\mathcal{R}_e$ whereby agreement is reached immediately by the second round.

$$\mathcal{R}_t \stackrel{\text{def}}{=} \left\{ \begin{array}{ll|l} \Gamma \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] & , \Gamma \triangleright \mathsf{cons}^n \mid \mathsf{req}^n\langle t\rangle & \\ \Gamma \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_j \mid \prod_{i=1}^n \mathsf{Pnt}_i(r_i, t)) & \begin{array}{l} 1 \le j \le n+1 \\ 1 \le r_i \le n+1 \end{array} \\ \Gamma \triangleright k_0[\![\mathbf{0}]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_{\text{done}} \mid \prod_{i=1}^n \mathsf{Pnt}_i(n+1, t)) & \end{array} \right\}$$

$$\mathcal{R}_f \stackrel{\text{def}}{=} \left\{ \begin{array}{ll|l} \Gamma \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] & , \Gamma \triangleright \mathsf{cons}^n \mid \mathsf{req}^n\langle f\rangle & \\ \Gamma \triangleright k_0[\![\textsc{succ}!\langle\rangle]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_j \mid \prod_{i=1}^n \mathsf{Pnt}_i(r_i, f)) & \begin{array}{l} 1 \le j \le n+1 \\ 1 \le r_i \le n+1 \end{array} \\ \Gamma \triangleright k_0[\![\mathbf{0}]\!] & , \Gamma \triangleright (\nu\, c_{1..n})(\mathsf{Test}_{\text{done}} \mid \prod_{i=1}^n \mathsf{Pnt}_i(n+1, f)) & \end{array} \right\}$$

□

Lemma 5.6.1 showed that whenever the tests $\mathsf{req}^n\langle e\rangle$, $\mathsf{req}^n\langle t\rangle$ and $\mathsf{req}^n\langle e\rangle$ where placed in the context of cons^n, in a setting where no failure occurs, they always terminated. In this second part, Lemma 5.6.2 states that this behaviour is preserved up to $n-1$ failures, that may occur at any point during the execution. According to Definition 4.3.6, this amounts to showing that $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$ is *dynamically* fault-tolerant up-to $n-1$ failures.

Lemma 5.6.2 (Fault Tolerance). The configurations $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle$, $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle$ and $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle$ preserve their behaviour under any valid fault context F^{n-1}. Stated otherwise, they are dynamically fault tolerant up to $n-1$ failures.

Proof. According to Proposition 4.4.3, Corollary 4.4.4 and Proposition 4.4.11, this amounts to showing that

1. $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle \;\preceq_\beta^{n-1}\; \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle$
2. $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle \;\preceq_\beta^{n-1}\; \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle$
3. $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle \;\preceq_\beta^{n-1}\; \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle$

More prosaically, we need to give three witness $n-1$ fault-tolerant simulation relations up-to β-moves for the configurations $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle$, $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle$ and $\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle e\rangle$. In the discussion of these witness relations, we refer to the ith participant of the configuration on the left of the relation as the ith *left hand participant* and call its right hand side equivalent the ith *right hand participant*; to distinguish between the two, we use the dashed notation, $\mathsf{Pnt}'_i(r'_i, v)$, for the ith right hand participant. The dashed notation is also used for the Test code on the right hand side configuration of the relations.

The $n-1$ fault tolerant simulation relation up-to β-moves satisfying clause 1 is $\mathcal{R}_t$ defined below:

$$\mathcal{R}_t \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle \quad , \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle t\rangle & \\ \\ \Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_j | \prod_{i=1}^{n} \mathsf{Pnt}_i(r_i, t) & \left| \begin{array}{l} 1 \leq j \leq h \leq n+1 \\ \text{if } \Gamma' \vdash l_i : \textbf{alive} \\ \quad \text{then } \mathsf{Pnt}_i(r_i, t)\, \mathcal{R}_{\text{Id}}\, N_i \\ \quad \text{else } \mathsf{Pnt}_i(r_i, t)\, \mathcal{R}_{\mathbf{0}}\, N_i \end{array} \right. \\ \quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_h | \prod_{i=1}^{n} N_i & \\ \\ \Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{\text{done}} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, t) & \left| \begin{array}{l} \text{if } \Gamma' \vdash l_i : \textbf{alive} \\ \quad \text{then } \mathsf{Pnt}_i(n+1, t)\, \mathcal{R}_{\text{Id}}\, N_i \\ \quad \text{else } \mathsf{Pnt}_i(n+1, t)\, \mathcal{R}_{\mathbf{0}}\, N_i \end{array} \right. \\ \quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{\text{done}} | \prod_{i=1}^{n} N_i & \end{array} \right\}$$

where

$$\mathcal{R}_{\text{Id}} = \left\{ \begin{array}{lll} \mathsf{Pnt}_i(0) & , \mathsf{Pnt}'_i(0) & \\ \mathsf{Pnt}_i(r, v) & , \mathsf{Pnt}'_i(r, v) & | \; v \in \{t, f\} \end{array} \right\}$$

$$\mathcal{R}_{\mathbf{0}} = \left\{ \; \mathsf{Pnt}_i(r, v) \quad , l_i[\![\mathbf{0}]\!] \;\middle|\; v \in \{t, f\} \; \right\}$$

In essence, $\mathcal{R}_t$ gives a mapping between the state of every $\mathsf{Pnt}_i(r, v)$ in a failure free setting and the corresponding i^{th} participant in a setting where it may fail. The relation states that as long as l_i is alive on the right hand side, its round number and value estimate must match that of its corresponding participant on the left hand side. This mapping is formalised in the relation $\mathcal{R}_{\text{Id}}$, ranging over systems limited to a single location, relating participants at the same round and with the same estimate, while abstracting away from differing peripheral code. If, on the other hand, l_i is dead on the right hand side, then we exploit up-to β moves, and use the structural rule (bs-dead) to map to it to the null process at l_i for simplification; this mapping is formalised in the relation $\mathcal{R}_{\mathbf{0}}$, again ranging over systems limited to a single location.

We again note that in this relation, $\mathcal{R}_t$, and the ones following it, participants progress in lockstep from one round to another. More specifically, P_i^r needs to receive n estimates for round r to proceed to round $r + 1$; in turn, the other participants P_j cannot sent their estimate for round r unless they are in round r and because of this interdependency, the round numbers of live participants one either side differ by at most 1. As we already discussed at the end of § 5.5, most of these τ-actions are abstracted away through our notation of aggregate states. Intuitively though, a participant on the *left* hand side can always progress to the next round with a t estimate in a number of τ-moves since all the participants are alive and can therefore send it their t estimate for that round. Moreover, a participant on the *right* hand side can always progress to the next round with a t estimate in a number of τ-moves as well: the sibling live participants can send their t estimate for that round whereas the monitors generate the remaining t estimates for that round for the sibling participants that are dead.

More specifically, in $\mathcal{R}_t$, a τ-action by a participant on either left or right configurations is matched by the identical τ-action by its dual, unless its is dead, in which case it is mapped by the empty transition. A fail-action on the right hand side is similarly matched by an empty transition on the left hand side. As we already stated, a dead participant does not prohibit any of its live sibling participants to match actions on the left hand side: its estimate is immediately substituted by the respective monitors, that release a t estimate for that round as well. In fact, since we work at up-to β-moves, when a location fails, we automatically transition to a state, not explicitly represented by our aggregate states, where all the active monitors on that location trigger and release their estimate, using the beta rule (b-mon).

The test system Test'_h on the right hand side is allowed to progress before that on the left hand side: its failure detectors, which allow the test contexts to ignore the decision of a dead participant, automatically trigger when a participant fails one the right hand side, thereby allowing the test to proceed to the next participant. The test on the left hand side however, Test_j, still has to wait for its participant to decide before it progresses to the next one and thus we have the condition $1 \leq j \leq j' \leq n + 1$. We also note that since $\mathcal{R}_t$ is a relation up-to β-moves, we abstract away from the name matching performed by the tests; this is once again not explicitly represented by our aggregate states.

The $n - 1$-fault tolerant simulation relation up-to β-moves satisfying clause 2 is $\mathcal{R}_f$

given below:

$$\mathcal{R}_f \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle \quad , \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle & \\ & \\ \Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_j | \prod_{i=1}^{n} \mathsf{Pnt}_i(r_i, f) & 1 \le j \le h \le n+1 \\ \quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_h | \prod_{i=1}^{n} N_i & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\ & \quad \text{then } \mathsf{Pnt}_i(r_i, f)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\ & \quad \text{else } \mathsf{Pnt}_i(r_i, f)\, \mathcal{R}_{\mathbf{0}}\, N_i \\ & \\ \Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{\mathrm{done}} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, f) & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\ \quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{\mathrm{done}} | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\ & \quad \text{else } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathbf{0}}\, N_i \end{array} \right\}$$

where $\mathcal{R}_{\mathrm{Id}}$ and $\mathcal{R}_{\geq}$ are defined as above. The relation $\mathcal{R}_f$ is very similar to $\mathcal{R}_t$. The only point worth noting is that participants on the right-hand side (having an f estimate) are not prohibited from progressing to the next round with an f estimate by dead siblings: the monitors for the dead siblings will release t estimates. Our implementation makes sure that participants with an f estimates never decide but merely collect estimates: thus the value of estimates are released by the monitors is irrelevant and the only thing required by the participant to progress to the next round is that n estimates are released or received.

The relation satisfying clause 3, that is for the test $\mathsf{req}^n\langle e\rangle$, turns out to be slightly harder than $\mathcal{R}_t$ and $\mathcal{R}_f$: whereas, in the first two relations, the individual participants could only differ in one aspect, that is liveness, in clause 3, they may differ in another aspect, namely the current estimate. Referring back to the corrupted broadcast discussion of § 5.1, corrupted broadcast (caused by failure) may prohibit the participants on the right hand side configuration of a fault-tolerant simulation relation from reaching agreement by round 2, as opposed to the case of the failure-free setting of the participants on the left hand side configuration of the relation. Thus, since the participants may be initialised with different values, in round 1, the value estimates of the participants on either side may not match for rounds equal or greater than 2. Even if failure does not lead to a corrupted broadcast, and these participants reach agreement, they may reach agreement on a *different* value from the one agreed on the left hand side: this happens when all the participants initialised with f fail before they can send their estimate and influence the update of any of the participants initialised with t. As a result, the participants on the right hand side reach agreement at some round greater or equal to 2 on t, whereas the participants on the left hand side, which are not subject to any failure, reach an agreement on f.

Due to this new factor, we construct $\mathcal{R}_e$ using the following mapping:

$$
\mathcal{R}_e \stackrel{\text{def}}{=} \left\{
\begin{array}{l|l}
\multicolumn{2}{l}{\Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle \quad , \Gamma \triangleright \mathsf{cons}^n \,|\, \mathsf{req}^n\langle f\rangle} \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_j | \prod_{i=1}^{n} \mathsf{Pnt}_i(r_i, v_i) & 1 \le j \le h \le n+1,\ 0 \le r_i \le 1, \\
 & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_h | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(r_i, v_i)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(r_i, v_i)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_j | \prod_{i=1}^{n} \mathsf{Pnt}_i(r_i, t) & 1 \le j \le h \le n,\ 1 \le r_i \le n+1, \\
 & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_h | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(r_i, t)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(r_i, t)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_j | \prod_{i=1}^{n} \mathsf{Pnt}_i(r_i, f) & 1 \le j \le h \le n,\ 1 \le r_i \le n+1 \\
 & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_h | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(1, t)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\
 & \quad\quad \text{or } \mathsf{Pnt}_i(r_i, f)\, \mathcal{R}_{\mathrm{Id}}\, N_i \text{ or } \mathsf{Pnt}_i(r_i, f)\, \mathcal{R}_{=}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(r_i, f)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{n+1} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, t) & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{n+1} | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(n+1, t)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(n+1, t)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{n+1} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, f) & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{n+1} | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{n+1} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, f) & \text{case } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{n+1} | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{=}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{\mathrm{done}} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, t) & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{\mathrm{done}} | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(n+1, t)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(n+1, t)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{\mathrm{done}} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, f) & \text{if } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{\mathrm{done}} | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathrm{Id}}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathbf{0}}\, N_i \\
\Gamma \triangleright (\nu\, c_{1..n})\mathsf{Test}_{\mathrm{done}} | \prod_{i=1}^{n} \mathsf{Pnt}_i(n+1, f) & \text{case } \Gamma' \vdash l_i : \textbf{alive} \\
\quad , \Gamma' \triangleright (\nu\, c_{1..n})\mathsf{Test}'_{\mathrm{done}} | \prod_{i=1}^{n} N_i & \quad \text{then } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{=}\, N_i \\
 & \quad \text{else } \mathsf{Pnt}_i(n+1, f)\, \mathcal{R}_{\mathbf{0}}\, N_i
\end{array}
\right\}
$$

where $\mathcal{R}_{\text{Id}}$ and $\mathcal{R}_{\mathbf{0}}$ are defined above and $\mathcal{R}_{=}$ is defined below: it maps the ith participant with an f estimate at round r, to the ith participant with an t estimate at the same round r.

$$\mathcal{R}_{=} \;=\; \left\{ \;\mathsf{Pnt}_i(r, f) \quad , \mathsf{Pnt}_i(r, t)\; \right\}$$

The three key clauses in $\mathcal{R}_e$ are the second, third and fourth:

- the second clause captures a state where not all participants have been initialised. We thus use the relation $\mathcal{R}_{\text{Id}}$ to relate participant that are either uninitialised ($r = 0$), or in the first round ($r = 1$). In these two cases, the participants have not updated their value estimate and since not decision has been made yet, it could not be interfered with by failure: thus we can safely use $\mathcal{R}_{\text{Id}}$.
- the third clause is dedicated to the case where all the live participants on both sides have been initialised with a t estimate. Similar to relation $\mathcal{R}_t$ given earlier, every participant is related using $\mathcal{R}_{\text{Id}}$ if alive and $\mathcal{R}_{\mathbf{0}}$ otherwise.
- the fourth clause is dedicated to the case where some participants have been initialised by a f estimate. As we discussed earlier, we have to map participants at round 2 or greater from the left hand side configuration, all of which have an estimate of f, to participants from the right hand side configuration which may not have the same estimate due to failure. We thus relate individual participants either using $\mathcal{R}_{\text{Id}}$, if the estimate matches, or else $\mathcal{R}_{=}$, if the estimate does not; note that the round number always matches. As usual, if the right hand participant is dead, we use $\mathcal{R}_{\mathbf{0}}$.

The fifth, sixth and seventh cases in $\mathcal{R}_e$, makes it explicit that for the test code to reach $n+1$, then all the participant need to agree. However, in the case where some participants were initialised with an f, the right-hand side participants may agree on t instead of f, as is the case of the participants on the left-hand side. Thus the fifth case describes the case where both the left-hand participants and right-hand participants have agreed on t, the sixth case describes the case where both the left-hand participants and right-hand participants have agreed on f and the seventh case represent the case were the left-hand side participants agree on f whereas the right-hand side participants agree on t, which is why we use $\mathcal{R}_{=}$ to relate the live participants as opposed to $\mathcal{R}_{\text{Id}}$. Finally, the eight, ninth and tenth cases are just the respective direct derivatives of state five, six and seven after they produce the output action $k_0 : \text{succ}!\langle\rangle$.

We still have two show that $\mathcal{R}_e$ is closed under all possible actions. We here outline the main actions for the main cases of the relation:

- If we are in the first clause of the relation, then any τ action from either side is matched by an identical τ-action on the other side, transitioning to the second clause. If the right hand configuration performs a fail action, this is matched by the empty action and we transition to the second clause as well.
- Similarly, in the second clause, any τ action from either side is matched by an identical τ-action on the other side. We transition to the third or fourth clause only when all participants have been initialised: if they were all initialised to t, then we transition to the third clause whereas if some where initialise to f, we transition to the fourth clause. Once again, a right-hand side fail action, this is matched by the empty action, remaining in the same clause.

- Similar to the relation $\mathcal{R}_t$, participant τ-actions on either side are matched by identical τ-actions on the opposite side. The test code on the right hand side may be at an advance stage compared to the test on the left hand side, that is $j \leq h$, because of empty decisions generated for dead participants. As a result, if the left-hand test code produces a τ-action, then this can either be matched by an identical τ-actions to reach the same state or else be matched by the empty move if the right hand side is already in that state; if the right-hand test code produces a τ-action, then this can either be matched by an identical τ-actions if the left-hand test can match it or else the empty move if it cannot. Once the test code on other side reaches $n + 1$, the other matches this τ-action by (at most) a weak τ-action, transitioning to case five in the relation.

- In the fourth clause, τ-actions by the left hand participants, can be matched by either an identical τ-action by the respective right-hand participant if it is alive, or else an empty move; in doing so the right-hand participant may now be related to the left hand participant using $\mathcal{R}_=$ instead of $\mathcal{R}_{\mathrm{Id}}$ if it transitions from a t estimate to an f estimate. Dually, a τ-action by a right-hand participant is always matched with a τ-action by the left participant. Right-hand side fail actions are once again matched by empty actions. Similar to the third case just described, as soon as one test code reaches $n + 1$, the other matches it and we transition to either case six, if the participants on the right-hand side agree on f, or else case seven, if they agree on t. We are guaranteed that they will agree because, for there to be a disagreement at round r, there must have been a corrupted broadcast for every previous round. Stated otherwise, if at round n, there is disagreement, there must have been $n - 1$ corrupted broadcasts, which mean that there have been $n - 1$ participants failing, meaning that there is only one participant left alive, and thus we automatically have agreement. □

As a result of these two Lemmas, we automatically have the proof of Proposition 5.4.2, which we here restate as a Corollary.

Corollary 5.6.3 (Test Termination for the Consensus-Solving Algorithm). For any consensus permitting network representation Γ for cons^n, then $\Gamma \triangleright \mathsf{cons}^n$ weakly terminates under any test $\mathsf{req}^n\langle v\rangle$ for $v \in \{t, f, e\}$. Stated otherwise:

$$
\begin{array}{l}
\forall \Gamma', N' \\
\quad \text{if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle t\rangle \,|\, \mathsf{F}^m \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0} \\
\quad \text{if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle f\rangle \,|\, \mathsf{F}^m \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0} \\
\quad \text{if } \Gamma \triangleright N \,|\, \mathsf{req}^n\langle e\rangle \,|\, \mathsf{F}^m \longrightarrow^* \Gamma' \triangleright N' \text{ then } \Gamma \triangleright N' \Downarrow_{\textsc{succ}@k_0}
\end{array}
$$

Proof. Immediate from Lemma 5.6.1 and Lemma 5.6.2. □

5.7 Summary

In this Chapter we used *co-induction techniques* to prove that the algorithm given in [CT96] which assumes perfect failure detectors, satisfies consensus. We opted for such bisimulation techniques instead of other techniques based on traces because the latter are insensitive to divergences and one of the conitions we set out to prove, namely strong termination, required that we show that the algorithm always yields a result.

We encoded the pseudo-code algorithm in our source language for Chapter 4, typed DπLoc; we also defined the consensus problem in terms of typed DπLoc code. For the actual correctness proof, we employed the definition of dynamic fault-tolerance to split the proof into two parts: in the first part, we isolated the behaviour of the system implemented in a failure free setting and prove that it satisfies the required conditions; in a second part we showed that the system is n–1 fault-tolerant meaning that this behaviour is maintained up-to n–1 faults. Splitting the proof into two phases simplified our analysis because:

- we only needed to consider the second phase, that is the fault tolerant behaviour of the algorithm under n1 dynamic failures if the first phases, which is considerably easier to prove, is successful.
- Our fault tolerance analysis is based on comparing the behaviour of our algorithm under n1 faults with the behaviour of the *same* algorithm under no faults. Here we exploited the common structure on both sides of the comparison in our analysis.

The up-to bisimulation techniques and fault-tolerant simulation relation developed earlier in Chapter 4 alleviated the burden of exhibiting witness bisimulation relations by abstracting over conflent moves and states, thus keeping the state space of the analysis manageable. We conjuncture that a similar method of analysis could be carried over with minimal effort, to prove the correctness of other distributed algorithms operating in the presence of failure; see [Lyn96, Tel94] for examples.

Correctness proofs for consensus-solving algorithms have been given already: in their pioneering work [CT96], Toueg and Chandra give proofs for the pseudo-code algorithms using algorithmic techniques. Lynch [Lyn96] formalises these algorithms using IO-automata so that correctness proofs can be given based on trace semantics. Nestmann, Fuzzatti and Merro , [NFM03], implemented the hardest of these algorithms (assuming only "eventually strong" failure detectors) in a tailor-made calculus with formal re-write rules, translated this formal syntax into an abstract interpretation consisting of matrices representing the state of the algorithm, and then proved correctness using the abstract interpretation. Palamidessi's work [Pal03] is also similar to ours, in that she encodes leader election algorithms in terms of process calculi. The aim of this work are however differ from ours and focusses on obtaining expressivity results for process calculi. The details of her work also differ from ours: the calculi considered do not express failures and thus there is no notion of fault tolerance; moreover, the proof techniques are based on graph transformations instead of bisimulations.

Chapter 6

Conclusion and Outlook

This thesis presented a study of system behaviour in the presence of failure. Our starting point was that of Hennessy and Riely, [RH01]; we extended this study of location failure to Dπ, [HR02, HMR04], a distributed π-calculus that prohibits communication across locations and can create new locations at runtime. We further extended this calculus to describe link failure as well as node failure, and studied the interplay of these two forms of failure on system behaviour. We then studied ways of guaranteeing dependable behaviour in this setting. Since faults are often hard to predict and prevent in distributed setting, we sought to achieve dependability through fault-tolerance, by introducing enough redundancy in our computation so as to withstand any abnormal behaviour cause by faults. Finally, we analysed a standard consensus-solving algorithm from the point of view of the redundancy it employs to satisfy the consensus criteria in the presence of failure.

6.1 Results obtained

Several results were obtained in this study, the most important of which can be found in Chapters 3 and 4. We here list these results not in the order of importance but rather in the order they were presented in this thesis.

- A *sound and complete theory* for a Dπ extension with *node failure.*
- A *sound and complete theory* for a Dπ extension with *node and link failure.*
- A formalisation of two *definitions of fault tolerance*, *static* and *dynamic* fault tolerance, together with *sound bisimulation techniques* for verifying fault tolerance.
- A demonstration of the *viability* of bisimulation and fault-tolerant simulation techniques for proving the *correctness of consensus-solving algorithms.*

6.2 Future Work

Our study is far from conclusive; rather, we believe it has paved the way for further lines of research in the field. Apart from the directions suggested at the end of the individual Chapters, we here discuss further research at greater length.

As we stated earlier in Chapter 3, dpiF is best viewed as a generic but succinct, well-founded framework from which numerous variations could be considered. For example links between sites could be uni-directional, rather than symmetric, or $\mathsf{ping}\ l.P\lceil Q\rceil$ could test for a *path* from the current site to l, rather than a direct connection. One could also limit the use of the fault inducing actions $\mathsf{kill} : l$ and $l \nleftrightarrow k$; for instance, disallowing them in the definition of the contextual equivalences would give a behavioural theory between systems running on *static* but possibly defective networks. More generally, one could allow the *recovery* of faults, in which dead nodes, or broken links may randomly be restored; transient faults are also directly related to issues such as *persistence* and *volatility* of code. Adapting our lts and the resulting bisimulation equivalence to such scenarios are in some cases straightforward, and in others, serious undertakings; a typical example of the former is the introduction of uni-directional links, while fault recovery and persistence would probably fall into the latter; higher-order theories of Dπ may need to be considered in the latter case.

The graph structure imposed on DπF locations should also be flexible enough to express other location structures as instances of the calculus. For instance, a hierarchical location structure such as that used in the distributed join-calculus can be elegantly encoded in DπF by imposing restrictions on the starting graph structure (it must be a tree) and the types of the new locations to be created (their only connection is to the parent). Moreover, by restricting the observer's view to the root nodes of this encoding, we can also encode the failure of a subtree as the breaking of the link connecting the root of the subtree to the remainder of the tree.

The obvious immediate extension to the work on fault tolerance, carried our in Chapter 4, is to extend the definitions and underlying theory to the full failure calculus, DπF. At a specification level, it would be interesting to investigate what constitutes a sensible definition of fault tolerance in the presence of two kinds of faults, node and link. For instance, one could decide to keep the unit count n to refer to nodes, and consider a completely disconnected node as a dead node; alternative, the definition may be extended to two parameters, n and m, where the n would denote the maximum node fault and m the maximum link faults that may be injected.

A more important line of research in the field of fault tolerance would however be to have a *compositional theory* of systems, allowing us to construct fault tolerant systems from smaller component sub-systems. Even though our work on fault-tolerant simulation relations can be a good starting point, dependent type systems, [YH02, HRY05], should shed more light on this problem whereas spatial logics such as [Rey02, ORY01] might yield a better understanding and treatment of redundancy. Having said this, the fault-tolerant definitions we formalised in this thesis should still serve as a valid specification for validating the resultant compositional theory for fault tolerance.

More generally, the framework adopted from [HR04], that is that of configurations, together with our adaptations, such as environments and types that encode notions of state, can be used to study other areas of distributed computing. The most obvious to us seems to be that of limited resource handling and garbage collection. Our current understanding is that similar concepts studied in our calculi can be found in such setting: for instance, the software *awareness* of the state of the surrounding resources, and mechanisms such as failure detection and fresh network discovery, could be translated to resource state detection and resource discovery. From the point of view of process calculi, resource management have already been studied in various literature; we here name a few of these which might be considered as suitable starting points[BBDS03, Hof02, IK05, Tel04, TZH02].

Finally, we believe that the application for the theory we developed in this thesis is a crucial next step. Through application, we are able to justify our design decisions and validate the results obtained. Moreover, any examples retrieved from these applications should motivate the immediate future research directions. We have stated earlier that DπF can be applied to the study of distributed software that needs to be aware of the *dynamic* computing context in which it is executing; various examples can be drawn from ad-hoc networks, embedded systems and generic routing software. In these settings, the software typically *discovers* new parts of the neighbouring network at runtime and *updates* its knowledge of the current underlying network with changes caused by failure. In addition, the theory of DπF married with a theory of fault tolerance should be adequate enough to give a lower level encoding of other preferred levels of abstraction in distributed systems, whose underlying implementations use fault tolerant routing techniques. Examples that springs to mind are the proposed middleware infrastructures such as [Wie02, MSW03] whereby nodes are organised as clusters or groups, with broadcast communication primitives extending to every member of the group.

Appendix A

Notation

Here we give the formal definitions for the various notation we have introduced for extracting information from network representations, and for updating them.

A.1 DπLoc Notation

Recall that for DπLoc a network repsresentation Π consists of the tuple $\langle \mathcal{N}, \mathcal{D} \rangle$, where $\mathcal{N}$ is a set of names known and $\mathcal{D}$ is the set of dead locations. We thus define the following judgements:

$$
\begin{array}{llll}
\Pi \vdash a:\mathtt{ch} & \stackrel{\text{def}}{=} & a \in \Pi_{\mathcal{N}} & \textit{(valid channels)} \\
\Pi \vdash l:\mathtt{loc[a]} & \stackrel{\text{def}}{=} & l \in \Pi_{\mathcal{N}} \ \wedge \ l \in \Pi_{\mathcal{A}} & \textit{(valid live location)} \\
\Pi \vdash l:\mathtt{loc[d]} & \stackrel{\text{def}}{=} & l \in \Pi_{\mathcal{N}} \ \wedge \ l \notin \Pi_{\mathcal{A}} & \textit{(valid dead location)} \\
\Pi \vdash l : \mathbf{alive} & \stackrel{\text{def}}{=} & \Pi \vdash l:\mathtt{loc[a]} & \textit{(live locations)} \\
\Pi \vdash k \leftarrow l & \stackrel{\text{def}}{=} & \Pi \vdash k:\mathbf{alive}, \ l:\mathbf{alive} & \textit{(k accessible from l)} \\
\Pi \vdash M & \stackrel{\text{def}}{=} & \mathbf{fn}(M) \in \Pi_{\mathcal{N}} & \textit{(valid systems)}
\end{array}
$$

We also define the following operations:

$$
\begin{array}{llll}
\Pi + a:\mathtt{ch} & \stackrel{\text{def}}{=} & \langle \Pi_{\mathcal{N}} \cup \{a\}, \Pi_{\mathcal{A}} \rangle & \textit{(adding fresh channel)} \\
\Pi + l:\mathtt{loc[a]} & \stackrel{\text{def}}{=} & \langle \Pi_{\mathcal{N}} \cup \{l\}, \Pi_{\mathcal{A}} \cup \{l\} \rangle & \textit{(adding fresh live location)} \\
\Pi + l:\mathtt{loc[d]} & \stackrel{\text{def}}{=} & \langle \Pi_{\mathcal{N}} \cup \{l\}, \Pi_{\mathcal{A}} \rangle & \textit{(adding fresh dead location)} \\
 & & & \\
\Pi - l & \stackrel{\text{def}}{=} & \begin{cases} \langle \Pi_{\mathcal{N}}, \Pi_{\mathcal{A}}/\{l\} \rangle & \text{if } l \in \Pi_{\mathcal{N}} \\ \Pi & \text{otherwise} \end{cases} & \textit{(killing a location)}
\end{array}
$$

A.2 DπF Notation

Network representations in DπF are based on the notion of linksets $\mathcal{L}$. We define the following operations and judgements, using a set of locations C:

$$\begin{array}{rcll}
\mathcal{L}/C & \stackrel{\text{def}}{=} & \{\langle k_1,k_2\rangle \mid \langle k_1,k_2\rangle \in \mathcal{L} \text{ and neither } k_1,k_2 \notin C\} & (\textit{filtering}) \\
\mathcal{L}\vdash k\leftarrow l & \stackrel{\text{def}}{=} & \langle l,k\rangle \in \mathcal{L} & (\textit{accessibility}) \\
\mathcal{L}\vdash k\leftarrow\!\!\!\rightsquigarrow l & \stackrel{\text{def}}{=} & \mathcal{L}\vdash k\leftarrow l \text{ or } \exists k'.\, \mathcal{L}\vdash k'\leftarrow l \text{ and } \mathcal{L}\vdash k\leftarrow\!\!\!\rightsquigarrow k' & (\textit{reachability}) \\
l\leftrightarrow C & \stackrel{\text{def}}{=} & \{l\leftrightarrow k \mid k\in C\} & (\textit{component creation}) \\
\mathcal{L}\leftarrow\!\!\!\rightsquigarrow l & \stackrel{\text{def}}{=} & \{k\leftrightarrow k' \mid k\leftrightarrow k' \in \mathcal{L} \text{ and } \mathcal{L}\vdash k\leftarrow\!\!\!\rightsquigarrow l\} & (\textit{component reference})
\end{array}$$

For DπF we have two kinds of network representations, ranged over by Δ and Σ. We define the following operations on them:

$$\begin{array}{rcll}
\Delta - l & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{A}}/\{l\},\ \Delta_{\mathcal{L}}\rangle & (\textit{location killing}) \\
\Sigma - l & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ \Sigma_{\mathcal{O}}/\{l\},\ \Sigma_{\mathcal{L}}/\{l\}\rangle & (\textit{location killing})
\end{array}$$

$$\begin{array}{rcll}
\Delta - l\leftrightarrow k & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{A}},\ \Delta_{\mathcal{L}}/\{\langle l,k\rangle,\langle k,l\rangle\}\rangle & (\textit{link breaking}) \\
\Sigma - l\leftrightarrow k & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ \Sigma_{\mathcal{O}}/\{\langle l,k\rangle,\langle k,l\rangle\},\ \Sigma_{\mathcal{L}}/\{\langle l,k\rangle,\langle k,l\rangle\}\rangle & (\textit{link breaking})
\end{array}$$

$$\begin{array}{rcll}
\Delta + a{:}\mathtt{ch} & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}}\cup\{a\},\ \Delta_{\mathcal{A}},\ \Sigma_{\mathcal{L}}\rangle & (\textit{adding a channel}) \\
\Sigma + a{:}\mathtt{ch} & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}}\cup\{a\},\ \Sigma_{\mathcal{O}},\ \Sigma_{\mathcal{H}}\rangle & (\textit{adding a channel})
\end{array}$$

$$\begin{array}{rcll}
\Delta + l{:}\mathtt{loc[a,C]} & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}}\cup\{l\},\ \Delta_{\mathcal{A}}\cup\{l\},\ \Sigma_{\mathcal{L}}\cup l\leftrightarrow \mathtt{C}\rangle & (\textit{adding a location}) \\
\Delta + l{:}\mathtt{loc[d,C]} & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}}\cup\{l\},\ \Delta_{\mathcal{A}},\ \Sigma_{\mathcal{L}}\cup l\leftrightarrow \mathtt{C}\rangle & \\
\Sigma + l{:}\mathtt{loc[d,C]} & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}}\cup\{l\},\ \Sigma_{\mathcal{O}},\ \Sigma_{\mathcal{H}}\rangle & (\textit{adding a location}) \\
\Sigma + l{:}\mathtt{loc[a,C]} & \stackrel{\text{def}}{=} & &
\end{array}$$

$$\begin{array}{lll}
\text{Case} & \mathtt{C}\cap\mathbf{dom}(\Sigma_{\mathcal{O}}) = \emptyset & \text{then } \langle \Sigma_{\mathcal{N}}\cup\{n\},\ \Sigma_{\mathcal{O}},\ \mathcal{H}'\rangle \\
 & & \quad \text{where: } \mathcal{H}' = \Sigma_{\mathcal{H}}\cup(l\leftrightarrow \mathtt{C}) \\
 & \mathtt{C}\cap\mathbf{dom}(\Sigma_{\mathcal{O}}) \neq \emptyset & \text{then } \langle \Sigma_{\mathcal{N}}\cup\{n\},\ \mathcal{O}',\ \mathcal{H}'\rangle \\
 & & \quad \text{where: } \mathcal{O}' = \Sigma_{\mathcal{O}}\cup(l\leftrightarrow \mathtt{C})\cup(\Sigma_{\mathcal{H}}\leftarrow\!\!\!\rightsquigarrow \mathtt{C}) \\
 & & \qquad \text{and } \mathcal{H}' = \Sigma_{\mathcal{H}}/(\Sigma_{\mathcal{H}}\leftarrow\!\!\!\rightsquigarrow \mathtt{C})
\end{array}$$

We next define translations from one network representation to the other, together with the definition of the observer network knowledge for every representation.

$$\begin{array}{rcll}
\Sigma(\Delta) & \stackrel{\text{def}}{=} & \langle \Delta_{\mathcal{N}},\ \Delta_{\mathcal{A}},\ \emptyset\rangle & (\textit{from } \Delta \textit{ to } \Sigma) \\
\Delta(\Sigma) & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ (\mathbf{loc}(\Sigma_{\mathcal{N}})/\mathbf{dom}(\Sigma_{\mathcal{O}}\cup\Sigma_{\mathcal{H}})),\ \Sigma_{\mathcal{O}}\cup\Sigma_{\mathcal{H}}\rangle & (\textit{from } \Sigma \textit{ to } \Delta) \\
\mathcal{I}(\Sigma) & \stackrel{\text{def}}{=} & \langle \Sigma_{\mathcal{N}},\ \Sigma_{\mathcal{O}}\rangle & (\textit{observer knowledge}) \\
\mathcal{I}(\Delta) & \stackrel{\text{def}}{=} & \mathcal{I}(\Sigma(\Delta)) &
\end{array}$$

Finally, we define judgements made using the various network representations. Ideally we would like that distinct network representations that have the same semantic interpretations yield the same judgements as shown below.

$$
\begin{array}{rcll}
\Sigma \vdash l : \textbf{alive} & \stackrel{\text{def}}{=} & l \in \textbf{dom}(\Sigma_{O} \cup \Sigma_{\mathcal{H}}) & (\textit{live locations}) \\
\Sigma \vdash l \leftrightarrow k & \stackrel{\text{def}}{=} & l \leftrightarrow k \in \Sigma_{O} \cup \Sigma_{\mathcal{H}} & (\textit{live link}) \\
\Sigma \vdash \mathtt{T} & \stackrel{\text{def}}{=} & \textbf{fn}(\mathtt{T}) \subseteq \Sigma_{\mathcal{N}} & (\textit{valid types}) \\
\Sigma \vdash n : \mathtt{T}, \tilde{n} : \tilde{\mathtt{T}} & \stackrel{\text{def}}{=} & \Sigma \vdash \mathtt{T} \text{ and } \Sigma + n : \mathtt{T} \vdash \tilde{n} : \tilde{\mathtt{T}} & \\
\Sigma \vdash N & \stackrel{\text{def}}{=} & \textbf{fn}(N) \subseteq \Sigma_{\mathcal{N}} & (\textit{valid systems}) \\
\Sigma \vdash k \leftarrow l & \stackrel{\text{def}}{=} & \Sigma_{O} \vdash k \leftarrow l \text{ or } \Sigma_{\mathcal{H}} \vdash k \leftarrow l & (\textit{accessibility}) \\
\Sigma \vdash k \leftarrow\!\!\!\sim l & \stackrel{\text{def}}{=} & \Sigma_{O} \vdash k \leftarrow\!\!\!\sim l \text{ or } \Sigma_{\mathcal{H}} \vdash k \leftarrow\!\!\!\sim l & (\textit{reachability})
\end{array}
$$

$$
\Delta \vdash l : \textbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N \quad \stackrel{\text{def}}{=} \quad \Sigma(\Delta) \vdash l : \textbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N
$$

$$
\begin{array}{rcll}
\mathcal{I} + n : \mathtt{L} & \stackrel{\text{def}}{=} & \langle \mathcal{I}_{\mathcal{N}} \cup \{n\},\ \mathcal{I}_{O} \cup \mathtt{L} \rangle & (\textit{updates}) \\
\mathcal{I} \vdash l : \textbf{alive} & \stackrel{\text{def}}{=} & l \in \textbf{dom}(\mathcal{I}_{O}) & (\textit{live locations}) \\
\mathcal{I} \vdash l \leftrightarrow k & \stackrel{\text{def}}{=} & l \leftrightarrow k \in \mathcal{I}_{O} & (\textit{live link}) \\
\mathcal{I} \vdash \mathtt{T} & \stackrel{\text{def}}{=} & \textbf{fn}(\mathtt{T}) \subseteq \textbf{dom}(\mathcal{I}_{O}) & (\textit{valid types}) \\
\mathcal{I} \vdash l[\![P]\!] & \stackrel{\text{def}}{=} & \textbf{fn}(P) \subseteq \mathcal{I}_{\mathcal{N}} \text{ and } l \in \textbf{dom}(\mathcal{I}_{O}) & (\textit{valid systems}) \\
\mathcal{I} \vdash (\nu\, n : \mathtt{T})N & \stackrel{\text{def}}{=} & \mathcal{I} \vdash \mathtt{T} \text{ and } \mathcal{I} + n : \mathtt{T} \vdash N & \\
\mathcal{I} \vdash N|M & \stackrel{\text{def}}{=} & \mathcal{I} \vdash N \text{ and } \mathcal{I} \vdash M &
\end{array}
$$

$$
\begin{array}{rcll}
\Delta \vdash_{\mathsf{obs}} l : \textbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N & \stackrel{\text{def}}{=} & \mathcal{I}(\Delta) \vdash l : \textbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N & (\textit{external judgments}) \\
\Sigma \vdash_{\mathsf{obs}} l : \textbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N & \stackrel{\text{def}}{=} & \mathcal{I}(\Sigma) \vdash l : \textbf{alive},\ l \leftrightarrow k,\ \mathtt{T},\ N &
\end{array}
$$

Finally we outline a number of operations on types used in reduction rules and transition rules.

$$
\begin{array}{rcll}
\mathtt{ch}/\{l_1, .., l_n\} & \stackrel{\text{def}}{=} & \mathtt{ch} & (\textit{type filtering}) \\
\mathtt{loc[C]}/\{l_1, .., l_n\} & \stackrel{\text{def}}{=} & \mathtt{loc[C}/\{l_1, .., l_n\}] &
\end{array}
$$

$$
\begin{array}{rcll}
\mathsf{inst}(\mathtt{loc[C]}, l, \Delta) & \stackrel{\text{def}}{=} & \mathtt{loc}[\{k \mid k \in \mathtt{C} \text{ and } \Delta \vdash k \leftarrow\!\!\!\sim l\}] & (\textit{instantiate}) \\
\mathsf{inst}(\mathtt{loc[C]}, l, \Sigma) & \stackrel{\text{def}}{=} & \mathtt{loc}[\{k \mid k \in \mathtt{C} \text{ and } \Sigma \vdash k \leftarrow\!\!\!\sim l\}] &
\end{array}
$$

$$
\mathsf{lnk}(n : \mathtt{T}, \Sigma) \quad \stackrel{\text{def}}{=} \quad \begin{array}{l} (n \leftrightarrow \mathtt{C}) \cup (\Sigma_{\mathcal{H}} \leftarrow\!\!\!\sim \mathtt{C}) \\ \quad \text{if } \mathtt{T} = \mathtt{loc[a, C]} \text{ and } \mathtt{C} \cap \textbf{loc}(\Sigma_{O}) \neq \emptyset \\ \emptyset \ \ \text{otherwise} \end{array} \qquad (\textit{link types})
$$

A.3 Typed DπLoc Notation

Recall that for typed DπLoc a network repsresentation Γ consists of the tuple $\langle \mathcal{T}, \mathcal{D} \rangle$, where $\mathcal{T}$ is a set of tuples, $n : \mathtt{W}$, matching defined names to their stateless type, and $\mathcal{D}$ is the set of dead locations. We thus define the following judgements:

$$
\begin{array}{llll}
\Gamma \vdash a{:}\mathtt{B}\langle \tilde{\mathtt{W}} \rangle & \stackrel{\text{def}}{=} & a{:}\mathtt{B}\langle \tilde{\mathtt{W}} \rangle \in \Gamma_{\mathcal{T}} & \textit{(channel judgement)} \\
\Gamma \vdash l{:}\mathtt{loc[B,d]} & \stackrel{\text{def}}{=} & l{:}\mathtt{loc[B]} \in \Gamma_{\mathcal{T}} \ \wedge \ l \notin \Gamma_{\mathcal{A}} & \textit{(dead location judgement)} \\
\Gamma \vdash l{:}\mathtt{loc[B,a]} & \stackrel{\text{def}}{=} & l{:}\mathtt{loc[B]} \in \Gamma_{\mathcal{T}} \ \wedge \ l \in \Gamma_{\mathcal{A}} & \textit{(live location judgement)} \\
\Gamma \vdash l : \textbf{alive} & \stackrel{\text{def}}{=} & \Gamma \vdash l{:}\mathtt{loc[B,a]} & \textit{(live locations)} \\
\Gamma \vdash k \leftarrow l & \stackrel{\text{def}}{=} & \Gamma \vdash k{:}\textbf{alive},\, l{:}\textbf{alive} & \textit{(k accessible from l)}
\end{array}
$$

We also define the following operations:

$$
\begin{array}{llll}
\Gamma + a{:}\mathtt{B}\langle \tilde{\mathtt{W}} \rangle & \stackrel{\text{def}}{=} & \langle \Gamma_{\mathcal{T}} \cup \{a{:}\mathtt{B}\langle \tilde{\mathtt{W}} \rangle\}, \Gamma_{\mathcal{A}} \rangle & \textit{(adding fresh channel)} \\
\Gamma + l{:}\mathtt{loc[B,a]} & \stackrel{\text{def}}{=} & \langle \Gamma_{\mathcal{T}} \cup \{l{:}\mathtt{loc[B]}\}, \Gamma_{\mathcal{A}} \cup \{l\} \rangle & \textit{(adding fresh live location)} \\
\Gamma + l{:}\mathtt{loc[B,d]} & \stackrel{\text{def}}{=} & \langle \Gamma_{\mathcal{T}} \cup \{l{:}\mathtt{loc[B]}\}, \Gamma_{\mathcal{A}} \rangle & \textit{(adding fresh dead location)} \\
\\
\Pi - l & \stackrel{\text{def}}{=} & \begin{cases} \langle \Pi_{\mathcal{N}}, \Pi_{\mathcal{A}}/\{l\} \rangle & \text{if } l \in \Pi_{\mathcal{N}} \\ \Pi & \text{otherwise} \end{cases} & \textit{(killing a location)} \\
\textbf{pub}(\Gamma) & \stackrel{\text{def}}{=} & \left\langle \left\{ n{:}\mathtt{T} \;\middle|\; \begin{array}{l} \Gamma \vdash n{:}\mathtt{T} \text{ and} \\ \mathtt{T} = \mathtt{p[A]} \text{ or } \mathtt{p}\langle \tilde{\mathtt{P}} \rangle \end{array} \right\}, \{ l \mid \Gamma \vdash l{:}\mathtt{p[a]} \} \right\rangle & \textit{(public network)} \\
\Gamma \vdash_{\mathsf{obs}} N, n & \stackrel{\text{def}}{=} & \textbf{pub}(\Gamma) \vdash N, n &
\end{array}
$$

A.4 Auxilliary Proofs

We here prove a lemma that is used to show that our lts of § 3.3 is closed over valid effective configurations.

Lemma A.4.1 (Valid Effective Network Updates). If Σ is a valid effective network, n is fresh in Σ and the type $\mathtt{T}$ is a valid type with respect to Σ, denoted as $\Sigma \vdash \mathtt{T}$ (see Appendix for definition) then $\Sigma + n\!:\!\mathtt{T}$ is a valid effective network.

Proof. The cases where $\mathtt{T} = \mathtt{ch}$ and $\mathtt{T} = \mathtt{loc[d, C]}$ are trivial so we focus our attention to the case where $\mathtt{T} = \mathtt{loc[d, C]}$; at this point, according to Definition 3.3.5, we have two possible subcases:

- If $\mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) = \emptyset$ then $\Sigma + n : \mathtt{loc[a, C]}$ has the form $\langle \Sigma_{\mathcal{N}} \cup \{n\},\ \Sigma_{\mathcal{O}},\ \mathcal{H}' \rangle$ where $\mathcal{H}' = \Sigma_{\mathcal{H}} \cup (l \leftrightarrow \mathtt{C})$. To prove that this resultant network is a valid effective network, we have to show that it adheres to the three consistency requirements, defined earlier in Definition 3.3.2:
    1. $\mathbf{dom}(\Sigma_{\mathcal{O}}) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$. This is immediate from the fact that Σ is valid and thus $\mathbf{dom}(\Sigma_{\mathcal{O}}) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}})$.
    2. $\mathbf{dom}(\mathcal{H}') \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$ and that $\mathcal{H}'$ is a linkset. The inclusion is obtained from the fact that $\mathbf{dom}(\Sigma_{\mathcal{H}}) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}})$ and the assumption that $\mathbf{loc}(\mathtt{loc[a, C]}) \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}})$. The fact that $\mathcal{H}' = \Sigma_{\mathcal{H}} \cup l \leftrightarrow \mathtt{C}$ is a linkset is immediate from the fact that $l \leftrightarrow \mathtt{C}$ is a component.
    3. $\mathbf{dom}(\Sigma_{\mathcal{O}}) \cap \mathbf{dom}(\mathcal{H}') = \emptyset$. This is immediately obtained from the assumptions that $\mathbf{dom}(\Sigma_{\mathcal{O}}) \cap \mathbf{dom}(\Sigma_{\mathcal{H}}) = \emptyset$, $n \notin \Sigma_{\mathcal{N}}$ and the condition for this subcase, that is $\mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) = \emptyset$.
- If $(\mathtt{C} \cap \mathbf{dom}(\Sigma_{\mathcal{O}}) \neq \emptyset)$ then $\Sigma + n : \mathtt{loc[a, C]}$ has the form $\langle \Sigma_{\mathcal{N}} \cup \{n\},\ \mathcal{O}',\ \mathcal{H}' \rangle$ where $\mathcal{O}' = \Sigma_{\mathcal{O}} \cup (l \leftrightarrow \mathtt{C} \cup (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C})$ and $\mathcal{H}' = \Sigma_{\mathcal{H}} / (\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C})$. One again, we have to prove that $\Sigma + n\!:\!\mathtt{loc[a, C]}$ satisfies the three consistency conditions:
    1. $\mathbf{dom}(\mathcal{O}') \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$ and that $\mathcal{O}'$ is a linkset. The proof here progresses similar to the second requirement of the previous subcase.
    2. $\mathbf{dom}(\mathcal{H}') \subseteq \mathbf{loc}(\Sigma_{\mathcal{N}} \cup \{n\})$ and that $\mathcal{H}'$ is a linkset. The proof for the inclusion is a simpler version of the above subcases, while the requirement that $\mathcal{H}' = \Sigma_{\mathcal{H}} / \Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}$ is a linkset is obtained from the fact that $\Sigma_{\mathcal{H}} \leftrightsquigarrow \mathtt{C}$ is a component and Lemma 3.3.4.
    3. $\mathbf{dom}(\mathcal{O}') \cap \mathbf{dom}(\mathcal{H}') = \emptyset$. This is obtained from the assumptions that $\mathbf{dom}(\Sigma_{\mathcal{O}}) \cap \mathbf{dom}(\Sigma_{\mathcal{H}}) = \emptyset$, $n \notin \Sigma_{\mathcal{N}}$ and the structure of $\mathcal{O}'$ and $\mathcal{H}'$. □

Bibliography

[Ama97] Roberto M. Amadio. An asynchronous model of locality, failure, and process mobility. In D. Garlan and D. Le Métayer, editors, *Proceedings of the 2nd International Conference on Coordination Languages and Models (COORDINATION'97)*, volume 1282, pages 374–391, Berlin, Germany, 1997. Springer-Verlag.

[AP94] Roberto M. Amadio and Sanjiva Prasad. Localities and failures. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 14, 1994.

[BBDS03] F. Barbanera, M. Bugliesi, M. Dezani, and V. Sassone. A calculus of bounded capacities. In *Proceedings of Advances in Computing Science, 9th Asian Computing Science Conference, ASIAN'03*, volume 2896 of *LNCS*, pages 205–223. Springer, 2003.

[Ber04] Martin Berger. Basic theory of reduction congruence for two timed asynchronous π-calculi. In *Proc. CONCUR'04*, 2004.

[BNP99] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Basic observables for processes. *Inf. Comput.*, 149(1):77–98, 1999.

[BNP02] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Trace and testing equivalence on asynchronous processes. *Inf. Comput.*, 172(2):139–164, 2002.

[CDT01] George Coulouris, Jean Dollimore, and Kindberg Tim. *Distributed Systems: Concepts and Design*. International Computer Science. Addison Wesley, 3 edition, 2001.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science, Special Issue on Coordination*, 240(1):177–213, June 2000.

[Chr91] Flavin Christian. Understanding fault tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[CHR05] Alberto Ciaffaglione, Matthew Hennessy, and Julian Rathke. Proof methodologies for behavioural equivalence in Dπ. Technical Report 03/2005, University of Sussex, 2005.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[DGP05] R. De Nicola, D. Gorla, and R. Pugliese. Basic observables for a calculus for global computing. In L. Caires et al., editor, *Proc. of 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, volume 3580 of *LNCS*, pages 1226–1238. Springer, 2005.

[DNGP04] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic observables for a calulus for global computing. Technical report, Universita di Roma, "La Sapienza", 2004.

[FGLD96] Cedric Fournet, Georges Gonthier, Jean Jaques Levy, and Remy Didier. A calculus of mobile agents. *CONCUR 96*, LNCS 1119:406–421, August 1996.

[Fis83] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, pages 127–140. Springer-Verlag, 1983.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[GG89] R.J. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions (extended abstract). In A. Kreczmar and G. Mirkowska, editors, Proceedings 14th Symposium on *Mathematical Foundations of Computer Science,* MFCS '89, Porąbka-Kozubnik, Poland, August/September 1989, volume 379 of *lncs*, pages 237–248. Springer-Verlag, 1989.

[HB00] Kohei Honda and Martin Berger. The two-phase commitment protocol in an extended pi-calculus. In Luca Aceto and Björn Victor, editors, *EXPRESS00: 7th International Workshop on Expressiveness in Concurrency*, volume 39, pages 105–130, Amsterdam, The Netherlands, 2000. Elsevier.

[HMR04] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322:615–669, 2004.

[Hof02] Martin Hofmann. The strength of non-size increasing computation. In *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 260–269, New York, NY, USA, 2002. ACM Press.

[HR98] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. In Uwe Nestmann and Benjamin C. Pierce, editors, *HLCL98: High-Level Concurrent Languages (Nice, France, September 12, 1998)*, volume 16(3), pages 3–17. Elsevier Science Publishers, 1998.

[HR02] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[HR04] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14:651–684, 2004.

[HRY05] Matthew Hennessy, Julian Rathke, and Nobuko Yoshida. Safedpi: A language for controlling mobile code. *Acta Informatica*, 2005. To appear.

[HY95] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[IK05] Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. *ACM Trans. Program. Lang. Syst.*, 27(2):264–313, 2005.

[JR04] Alan Jeffrey and Julian Rathke. A theory of bisimulation for a fragment of concurrent ml with local names. *Theoretical Computer Science*, 2004.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[MAB⁺98] Martin W. Murhammer, Orcun Atakan, Stefan Bretz, Larry R. Pugh, Kazunari Suzuki, and David H. Wood. *TCP/IP Tutorial and Technical Overview*. IBM Redbooks. International Technical Support Organization, 6 edition, October 1998.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[MS92] Robin Milner and Davide Sangiorgi. Barbed bisimulation. *ICALP'92: Automata, Languages and Programming*, 632, 1992.

[MSW03] Sergio Mena, André Schiper, and Pawel T. Wojciechowski. A step towards a new generation of group communication systems. In Markus Endler and Douglas C. Schmidt, editors, *Middleware*, volume 2672 of *Lecture Notes in Computer Science*, pages 414–432. Springer, 2003.

[NFM03] Nestmann, Fuzzati, and Merro. Modeling consensus in a process calculus. In *CONCUR: 14th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2003.

[NH84] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

[ORY01] Peter O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proc. of CSL'01*, volume 2142 of *LNCS*, pages 1–19. Springer-Verlag, 2001.

[Pal03] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[Pra87] K. V. S. Prasad. *Combinators and Bisimulation Proofs for Restartable Systems*. PhD thesis, Department of Computer Science, University of Edinburgh, December 1987.

[Rey02] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS'02*, pages 55–74. IEEE Computer Society, 2002.

[RH01] James Riely and Matthew Hennessy. Distributed processes and location failures. *Theoretical Computer Science*, 226:693–735, 2001.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

[SS83] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *Computer Systems*, 1(3):222–238, 1983.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus*. Cambridge University Press, 2001.

[Tel94] Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, New York, NY, USA, 1994.

[Tel04] D. Teller. Recollecting resources in the pi-calculus. In *Proceedings of TCS 2004 (tbp)*, 2004.

[TZH02] D. Teller, P. Zimmer, and D. Hirschkoff. Using Ambients to Control Resources. In *Proc. of CONCUR'02*, volume 2421 of *LNCS*. Springer Verlag, 2002.

[VR01] Paulo Verissimo and Luis Rodrigues. *Distributed Systems for System Architects*. Kluwer Academic Publishers, 2001.

[Wie02] M. Wiesmann. *Group Communications and Database Replication: Techniques, Issues and Performance*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, May 2002. Number 2577.

[YH02] Nobuko Yoshida and Matthew Hennessy. Assigning types to processes. *Information and Computation*, 173:82–120, 2002.